ASSETBUNDLE FUNDAMENTALS

ASSeTBUNDLE基础

版本检查: 5.3

-

难度: 高级

*This is the fourth chapter in a*[*series of articles covering Assets, Resources and resource management*](http://unity3d.com/learn/tutorials/topics/best-practices/guide-asset-bundles-and-resources)*in Unity 5.*

*这里是Unity5中 series 关于Assets，Resources以及resource管理器的一系列文章中的第四章节*

This chapter discusses AssetBundles. It introduces the fundamental systems upon which AssetBundles are built, as well as the core APIs used to interact with AssetBundles. In particular, it discusses both the loading and unloading of AssetBundles themselves, as well as the loading and unloading of specific Asset and Objects from AssetBundles.

这一章节讨论了AssetBundles。它介绍了用于构建AssetBundle的基础系统，以及那些用来和AssetBundle交互的核心API。特别的是，它讨论了加载和卸载AssetBundle自身以及加载和卸载AssetBundle中的特殊资源和对象。

For more patterns and best practices on the uses of AssetBundles, see the next chapter in this series.

关于更多关于使用AssetBundle的模式以及最好的用力，参见这个系列的下一章节。

3.1. OVERVIEW

概要

The AssetBundle system provides a method for storing one or more files in an archival format that Unity can index. The purpose of the system is to provide a data delivery method compatible with Unity's serialization system. AssetBundles are Unity's primary tool for the delivery and updating of non-code content after installation. This permits developers to reduce shipped asset size, minimize runtime memory pressure, and selectively load content that is optimized for the end-user's device.

AssetBundle系统提供了一种以某个Unity可以识别的归档格式来存储一个或多个文件的方法。这个系统的目的就在于通过Unity的序列化系统提供一个数据传输的简易方式。AssetBundle是Unity在安装完成后传输以及更新非代码内容的主要工具。这使得开发者能够降低传输的资源大小，降低运行时内存压力以及可选择的加载用户终端上的经过优化的内容。

Understanding the way AssetBundles work is essential to building a successful Unity project for mobile devices.

理解AssetBundle的工作方式是构建一个成功的移动平台Unity项目最重要的部分。

## 3.2. WHAT'S IN AN ASSETBUNDLE?

An AssetBundle consists of two parts: a header and a data segment.

一个AssetBundle包括了两个部分：头部分以及数据部分

The header is generated by Unity when the AssetBundle is built. It contains information about the AssetBundle, such as the AssetBundle's identifier, whether the AssetBundle is compressed or uncompressed, and a manifest.

头部分是由Unity在AssetBundle构建时生成的。它包含了AssetBundle的信息例如AssetBundle的id，是否压缩，以及一个表单（manifest）

The manifest is a lookup table keyed by an Object's name. Each entry provides a byte index that indicates where a given Object can be found within the AssetBundle's data segment. On most platforms, this lookup table is implemented as an STL std::multimap. While the specific algorithm used by any given platform's implementation of the STL varies, most are a variety of balanced search tree. Windows and OSX-derived platforms (including iOS) employ a red-black tree. Therefore, the time needed to construct the manifest will increase more than linearly as the number of Assets within an AssetBundle grows.

这个表单是一个用于通过对象的名称进行查询的表格。每个条目提供了一个字节的索引，表示了一个给定对象在AssetBundle数据部分的位置。在大部分的平台上，这个查找表是使用STL std::multimap实现的，这会随着STL的变化在不同平台上实现具体的算法，类似于一个变化的平衡查找。Windows和OSX类的平台（包括iOS）使用的是红黑树。所以用于构建这个表单的时间会随着AssetBundle中的资源数目增长而呈线性增长。

The data segment contains the raw data generated by serializing the Assets in the AssetBundle. If the data segment is compressed, then the LZMA algorithm has been applied to the collective sequence of serialized bytes - that is, all assets are serialized, and then the complete byte array is compressed.

数据部分包含了由AssetBundle中序列化了的资源行数据。如果数据部分是压缩的，那么LZMA算法就用于这些序列化字节序列集。意味着所有的资源都是序列化的，完整的字节数组也是压缩的。

Prior to Unity 5.3, Objects could not be compressed individually inside an AssetBundle. As a consequence, if a version of Unity before 5.3 is instructed to read one or more Objects from a compressed AssetBundle, Unity had to decompress the entire AssetBundle. Generally, Unity cached a decompressed copy of the AssetBundle to improve loading performance for subsequent loading requests on the same AssetBundle.

在Unity5.3之前，在AssetBundle中的对象不能单独的压缩。因此，如果一个Unity5.3之前版本被命令去读取压缩的AssetBundle中的一个或多个对象，Unity需要先将AssetBundle整个进行解压缩。通常情况下，Unity将解压缩的AssetBundle进行缓存来提升之后从同一个AssetBundle中加载的性能。

Unity 5.3 added a LZ4 compression option. AssetBundles built with the LZ4 compression option will compress individual Objects within the AssetBundle, allowing Unity to store compressed AssetBundles on disk. This also allows Unity to decompress individual Objects without needing to decompress the entire AssetBundle.

Unity5.3添加了LZ4压缩选项。通过LZ4压缩选项构建的AssetBundle会单独对AssetBundle中的每个对象进行压缩，运行Unity在磁盘上存储压缩的AssetBundle。这使得Unity可以针对每个独立的对象进行解压缩而不用解压整个AssetBundle。

## 3.3. THE ASSETBUNDLE MANAGER

ASSETBUNDLE管理器

Unity develops and maintains a reference implementation of an AssetBundle Manager on Bitbucket. This Manager employs many of the concepts and APIs detailed in this chapter, and provides a useful starting point for any project that must integrate AssetBundles into its resource-management workflow.

Unity在Bitbucket上开发并且维护了AssetBundle管理器的引用实现。这个管理器采用了许多整个章节中提到的概念和接口，并且为那些必须将AssetBundle整合进资源管理工作流程的项目起了一个好头。

Notable features include a "simulation mode". When active in the Unity Editor, this mode will transparently redirect requests for Assets tagged into AssetBundles to the original Assets within the project's /Assets/ folder. This allows developers to work on a project without needing to rebuild AssetBundles.

显著的功能包括了一个“模拟模式”。当打开Unity编辑器时，整个模式会隐式的将对AssetBundle中标记的资源的请求重定向到原始的资源目录“/Assets”中。这使得开发者在项目中工作时可以不用重新构建AssetBundle。

The AssetBundle Manager is an open-source project and can be found [here](https://bitbucket.org/Unity-Technologies/assetbundledemo).

AssetBundle管理器是一个开源的工程，在[这里](https://bitbucket.org/Unity-Technologies/assetbundledemo)可以查看。

## 3.4. LOADING ASSETBUNDLES

加载AssetBundle

In Unity 5, AssetBundles can be loaded via four distinct APIs. The behavior of these four APIs is different depending on two criteria:

在Unity5中，AssetBundle可以以4种方式进行加载。这4种接口的行为可以通过下面两条准则进行区分：

1. Whether the AssetBundle is LZMA compressed, LZ4 compressed or uncompressed

AssetBundle是LZMA压缩，LZ4压缩还是不压缩。

1. The platform on which the AssetBundle is being loaded

AssetBundle加载的平台

The four APIs in question are:

这4个接口是：

* [AssetBundle.LoadFromMemoryAsync](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromMemoryAsync.html)
* [AssetBundle.LoadFromFile](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromFile.html)
* [WWW.LoadFromCacheOrDownload](http://docs.unity3d.com/ScriptReference/WWW.LoadFromCacheOrDownload.html)
* [UnityWebRequest](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.html)'s [DownloadHandlerAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle.html) (on Unity 5.3 or newer)

[UnityWebRequest](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.html)'s [DownloadHandlerAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle.html) (在Unity 5.3或者更高的版本中)

### 3.4.1. AssetBundle.LoadFromMemoryAsync

**Unity's recommendation is not to use this API.**

**Unity建议不要使用这个接口**

Unity 5.3.3 Update: This API was renamed in Unity 5.3.3. In Unity 5.3.2 (or older), this API was known as***AssetBundle.CreateFromMemory***. Its functionality has not changed.

Unity5.3.3更新：这个API在Unity5.3.3中重新命名了。在Unity5.3.2（或者更早）中这个接口是***AssetBundle.CreateFromMemory*** 它的功能没有变化。

[AssetBundle.LoadFromMemoryAsync](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromMemoryAsync.html) loads an AssetBundle from a managed-code byte array (byte[] in C#). It will always copy the source data from the managed-code byte array into a newly-allocated, contiguous block of native memory. If the AssetBundle is LZMA compressed, it will decompress the AssetBundle while copying. Uncompressed and LZ4-compressed AssetBundles will be copied verbatim.

[AssetBundle.LoadFromMemoryAsync](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromMemoryAsync.html)通过一个托管代码的字节数组（c#中的byte[]）加载了一个AssetBundle。这总是会复制原始的数据，从托管代码的字节数组到新分配的，连续块状的本地内存。如果Assetb是LAMA压缩的，它会在拷贝时对AssetBundle进行解压。没有压缩的以及LZ4压缩的AssetBundle则是完全拷贝。

The peak amount of memory consumed by this API will be at least twice the size of the AssetBundle: one copy in native memory created by the API, and one copy in the managed byte array passed to the API. Assets loaded from an AssetBundle created via this API will therefore be duplicated three times in memory: once in the managed-code byte array, once in the native-memory copy of the AssetBundle and a third time in GPU or system memory for the asset itself.

这种加载方式的内存消耗峰值至少是AssetBundle大小的2倍：一份通过接口在本地内存中创建的以及一份传给接口的托管字节数组。通过这个接口从AssetBundle中加载的资源因此会在内存中重复3次：托管字节数组，AssetBundle的本地内存拷贝以及资源自身在GPU或者系统内存中占用的。

### 3.4.2. AssetBundle.LoadFromFile

Unity 5.3 update: This API was renamed in Unity 5.3. In Unity 5.2 (or older), this API was known as***AssetBundle.CreateFromFile***. Its functionality has not been changed.

Unity 5.3更新：这个API在Unity5.3中重新命名了。在Unity5.2（或者更早）中，这个接口是***AssetBundle.CreateFromFile.它的功能没有变化。***

[AssetBundle.LoadFromFile](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromFile.html) is a highly-efficient API intended for loading uncompressed AssetBundle from local storage, such as a hard disk or an SD card. If the AssetBundles are uncompressed or LZ4 compressed, the API will behave as follows:

[AssetBundle.LoadFromFile](http://docs.unity3d.com/ScriptReference/AssetBundle.LoadFromFile.html)是一个非常方便用于从本地空间（例如硬盘或者SD卡）加载没有压缩的AssetBundle的接口。如果AssetBundle是未压缩或者LZ4压缩的，。这个接口会表现如下：

Mobile devices: The API will only load the AssetBundle's header, and will leave the remaining data on disk. The AssetBundle's Objects will be loaded on-demand as loading methods (e.g. AssetBundle.Load) are called or as their InstanceIDs are dereferenced. No excess memory will be consumed in this scenario.

移动设备：这个接口只会加载AssetBundle的头部分，会把剩下的部分留在硬盘上。AssetBundle中的对象会在加载方法（如AssetBunle.Load）执行或者他们的实例ID被间接引用时按需加载。这这个方式中没有额外的内存消耗。

Unity Editor: The API will load the entire AssetBundle into memory, as if the bytes were read off disk and AssetBundle.LoadFromMemoryAsync was used. This API can cause memory spikes to appear during AssetBundle loading if the project is profiled in the Unity Editor. This should not affect performance on-device and these spikes should be re-tested on-device before taking remedial action.

Unity 编辑器：这个接口会加载整个AssetBundle到内存中，从磁盘上读取字节，就像AssetBundle.LoadFromMemoryAsync的用法那样。这个接口在AssetBundle加载时如果在Unity中开启了分析器就可以看到内存巨大的波动。这不会影响设备上的性能，而这些波动应该在采取补救措施之前在设备上重新测试。

Note: On Android devices with Unity 5.3 or older, this API will fail when trying to load AssetBundles from the Streaming Assets path. This is because the contents of that path will reside inside a compressed .jar file. For more details, see the section [Distribution - shipped with project](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Shipped_with_Project) section of the [AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns) chapter. This issue is resolved in Unity 5.4. Games built with Unity 5.4 or newer can now use this API to load Asset Bundles from Streaming Assets.

注意：Unity5.3或者更早的版本在Android设备上，这个接口在试图通过Streaming Assets path加载AssetBundle时会出错，这是因为那个路径下的内容在压缩的jar文件中。更多的信息，查看 [AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns)章节的 [Distribution - shipped with project](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Shipped_with_Project)部分

Note: Calls to AssetBundle.LoadFromFile will always fail for LZMA-compressed AssetBundles.

注意：对LZMA压缩的AssetBundle调用 AssetBundle.LoadFromFile 总是会失败的。

### 3.4.3. WWW.LoadFromCacheOrDownload

[WWW.LoadFromCacheOrDownload](http://docs.unity3d.com/ScriptReference/WWW.LoadFromCacheOrDownload.html) is a useful API for loading Objects both from remote servers and from local storage. Files can be loaded from local storage via a file:// URL. If the AssetBundle is present in the Unity cache, this API will behave exactly like AssetBundle.LoadFromFile.

[WWW.LoadFromCacheOrDownload](http://docs.unity3d.com/ScriptReference/WWW.LoadFromCacheOrDownload.html)是一个有效的能同时从本地空间和远程服务器上加载对象的接口。文件可以通过fille://URL从本地空间进行加载。如果AssetBundle当前在Unity的缓存中，这个接口就和AssetBundle.LoadFromFile一样。

If an AssetBundle has not yet been cached, then WWW.LoadFromCacheOrDownload will read the AssetBundle from its source. If the AssetBundle is compressed, it will be decompressed using a worker thread and written into the cache. Otherwise, it will be written directly into the cache via the worker thread.

如果一个AssetBundle还没有被缓存，那么[*WWW.LoadFromCacheOrDownload*会从它的来源读取AssetBundle。如果AssetBundle](http://WWW.LoadFromCacheOrDownload会从它的来源读取AssetBundle。如果AssetBundle)被压缩了，那么它会通过一个工作线程来解压缩并且写进缓存中。其他情况它都会通过工作线程直接写进缓存中。

Once the AssetBundle is cached, WWW.LoadFromCacheOrDownload will load header information from the cached, decompressed AssetBundle. The API will then behave identically to an AssetBundle loaded with AssetBundle.LoadFromFile.

当AssetBundle被缓存后，WWW.LoadFromCacheOrDownload 会从缓存中加载头信息，解压AssetBundle。这个接口和AssetBundle通过AssetBundle.LoadFromFile.加载下载后的字节一样。

Note: While the data will be decompressed and written to the cache via a fixed-size buffer, the WWW object will keep a full copy of the AssetBundle's bytes in native memory. This extra copy of the AssetBundle is kept to support the [WWW.bytesproperty](http://WWW.bytesproperty).

注意：当数据会被解压缩然后通过一个固定大小的缓冲区写进缓存时，WWW对象会在内存中保持AssetBundle字节的完整拷贝。这个AssetBundle的额外拷贝是用于支持[WWW.bytesproperty](http://WWW.bytesproperty)。

**Due to the memory overhead of caching an AssetBundle's bytes in the WWW object,** it is recommended that all developers using WWW.LoadFromCacheOrDownload ensure that their AssetBundles remain small - a few megabytes, at most. It is also recommended that developers operating on limited-memory platforms, such as mobile devices, ensure that their code downloads only a single AssetBundle at a time, in order to avoid memory spikes. For more discussion of AssetBundle sizing, see the [Asset assignment strategies](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Asset_Assignment_Strategies) section in the [AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns) chapter.

由于WWW对象缓存AssetBundle字节时的内存开销，建议所有使用WWW.LoadFromCacheOrDownload的开发者确保他们的AssetBundle最多保持少量mb大小。也建议所有在内存有限的平台的开发者，例如移动平台的开发者们，确保代码一次只下载一个AssetBundle，来避免内存波动。更多关于AssetBundle大小的讨论，参见 [AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns) 章节的 [Asset assignment strategies](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Asset_Assignment_Strategies) 部分。

Note: Each call to this API will spawn a new worker thread. Be careful of creating an excessive number of threads when calling this API multiple times. If more than 5-10 AssetBundles need to be downloaded, it is recommended that code be written to ensure that only a few AssetBundle downloads are running simultaneously.

注意：每次对这个接口的调用会开启一个新的工作线程。在多次调用这个接口时要小心创建过量的线程。如果有超过5-10个AssetBundle需要下载，建议代码能确保只有一部分AssetBundle的下载会同时执行。

### 3.4.4. AssetBundleDownloadHandler

Introduced on mobile platforms in Unity 5.3, the [UnityWebRequest](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.html) API provides a more flexible alternative to Unity's [WWW](http://docs.unity3d.com/ScriptReference/WWW.html)API. UnityWebRequest allows developers to specify exactly how Unity should handle downloaded data and allows developers to eliminate unnecessary memory usage. The simplest way to download an AssetBundle via UnityWebRequest is the [UnityWebRequest.GetAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.GetAssetBundle.html) API.

介绍Unity5.3上的移动平台，[UnityWebRequest](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.html) 接口提供了一个相对于[WWW](http://docs.unity3d.com/ScriptReference/WWW.html)更方便的接口。UnityWebRequest允许开发者标示Unity应该怎么处理下载后的数据以及允许开发者消除不必要的内存使用。通过UnityWebRequest下载AssetBundle最简单的方式就是 [UnityWebRequest.GetAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.UnityWebRequest.GetAssetBundle.html)接口。

For the purposes of this guide, the class of interest is [DownloadHandlerAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle.html). When used, this Download Handler behaves similarly to WWW.LoadFromCacheOrDownload. Using a worker thread, it streams downloaded data into a fixed-size buffer and then spools the buffered data to either temporary storage or the AssetBundle cache, depending on how the Download Handler has been configured. LZMA-compressed AssetBundles will be decompressed during download and cached uncompressed.

这个指南的目的在于，介绍[DownloadHandlerAssetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle.html)。让使用时，DownloadHandler的行为类似于WWW.LoadFromCacheOrDownload.使用一个工作线程，它将下载的数据传输进一个固定大小的缓冲区中然后把缓冲的数据存储进临时的存储空间或者AssetBundle缓存中，这取决于DownloadHandler是怎么配置的。LZMA压缩的AssetBundle会在下载过程中解压，然后以非压缩的方式进行缓存。

All of these operations occur in native code, eliminating the risk of expanding the managed heap. Further, this Download Handler does not keep a native-code copy of all downloaded bytes, further reducing the memory overhead of downloading an AssetBundle.

When the download is complete, the [assetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle-assetBundle.html) property of the Download Handler provides access to the downloaded AssetBundle, as if AssetBundle.LoadFromFile had been called on the downloaded AssetBundle.

所有的这些操作都发生在源代码中，消除了扩展托管堆的风险。此外，这个DownloadHandler不会保留所有下载字节的源代码副本，还降低了下载AssetBundle的内存消耗。当下载完成时，DownloadHandler的[assetBundle](http://docs.unity3d.com/ScriptReference/Experimental.Networking.DownloadHandlerAssetBundle-assetBundle.html) 属性提供了访问下载的AssetBundle的方式，类似于对下载的AssetBundle调用 AssetBundle.LoadFromFile 。

The UnityWebRequest API also supports caching in a manner identical to WWW.LoadFromCacheOrDownload. If caching information is provided to a UnityWebRequest object, and the requested AssetBundle already exists in Unity's cache, then the AssetBundle will become available immediately and this API will operate identically to AssetBundle.LoadFromFile.

UnityWebRequest接口也支持用指定的标示进行缓存，和 WWW.LoadFromCacheOrDownload.一样。如果缓存信息提供给UnityWebRequest对象，而请求的AssetBundle已经在Unity的缓存中了，那么AssetBundle会立即变成可用的，而这个接口操作就和 AssetBundle.LoadFromFile相同。

Note: The Unity AssetBundle cache is shared between WWW.LoadFromCacheOrDownload and UnityWebRequest. Any AssetBundle downloaded with one API will also be available via the other API.

注意：Unity AssetBundle缓存在WWW.LoadFromCacheOrDownload 和 UnityWebRequest.中是共享的，任意通过某个接口下载的AssetBundle都可以通过另一个接口进行访问。

Note: Unlike WWW, the UnityWebRequest system has an internal pool of worker threads and an internal job system to ensure that developers cannot start an excessive number of simultaneous downloads. The size of the thread pool is not currently configurable.

注意：和WWW不同，UnityWebRequest系统有一个内部的工作线程池以及一个内部的任务系统，来确保开发者不会同时开启过量的下载。线程池的大小目前是不可配置的。

**3.4.5. Recommendations**

**3.4.5. 建议**

In general, [AssetBundle.LoadFromFile](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#AssetBundle.LoadFromFile) should be used whenever possible. This API is the most efficient in terms of speed, disk usage and runtime memory usage.

通常情况下，应该尽可能的使用[AssetBundle.LoadFromFile](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#AssetBundle.LoadFromFile)。这个接口从速度，磁盘使用以及运行时内存都是最高效的。

For projects that must download or patch AssetBundles, it is strongly recommended to use [UnityWebRequest](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#AssetBundleDownloadHandler) for projects using Unity 5.3 or newer, and [WWW.LoadFromCacheOrDownload](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#WWW.LoadFromCacheOrDownload) for projects using Unity 5.2 or older. As detailed in the[Distribution](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Distribution) section of the next chapter, it is possible to prime the AssetBundle Cache with Bundles included within a project's installer.

对于那些必须下载或者打包AssetBundle的项目，强烈建议在Unity5.3或者更高版本中使用[UnityWebRequest](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#AssetBundleDownloadHandler) ，在Unity5.2或更老的版本中使用[WWW.LoadFromCacheOrDownload](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#WWW.LoadFromCacheOrDownload) 。下一章节的[Distribution](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Distribution)部分详细介绍了，它可以使得AssetBundle以包的形式进行缓存，即使是在一个已经安装好的程序上。

When using WWW.LoadFromCacheOrDownload, it is strongly recommended to ensure that the project's AssetBundles remain smaller than 2-3% of the project's maximum memory budget to prevent application termination due to memory-usage spikes. For most projects, AssetBundles should not exceed 5MB in file size, and no more than 1-2 AssetBundles should be downloaded simultaneously.

当使用[WWW.LoadFromCacheOrDownload](http://WWW.LoadFromCacheOrDownload)时，强烈建议确保项目的AssetBundle保持比项目最大内存预算的2-3%更小，来确保应用程序不会应该内存占用的波动而终止。对于大多数的项目，AssetBundle的文件大小不应该超过5MB，最多1-2个AssetBundle进行同时下载。

When using either WWW.LoadFromCacheOrDownload or UnityWebRequest, ensure that the downloader code properly callsDispose after loading the AssetBundle. Alternately, C#'s [using](https://msdn.microsoft.com/en-us/library/yh598w02.aspx) statement is the most convenient way to ensure that a WWWor UnityWebRequest is safely disposed.

当使用WWW.LoadFromCacheOrDownload 或者UnityWebRequest时，确保下载的代码在加载AssetBundle之后调用了Dispose。C#的[using](https://msdn.microsoft.com/en-us/library/yh598w02.aspx)声明是最方便的确保WWW或者UnityWebRequest 安全释放的方式。

For projects with substantial engineering teams and unique caching or downloading requirements, a custom downloader may be necessary. Writing a custom downloader is a non-trivial engineering task, and any custom downloader should be made compatible with AssetBundle.LoadFromFile. See the [Distribution](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Distribution) section of the next chapter for more details.

对于那些有着大量工程团队以及独特的缓存或者下载需求的项目而言，可能需要一个自定义的下载器。撰写一个自定义的下载器是一个不简单的工程任务，任何自定义的下载器都应该和AssetBundle.LoadFromFile 兼容。更多详细的内容参见下一章的[Distribution](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Distribution)部分。

## 3.5. LOADING ASSETS FROM ASSETBUNDLES

从ASSETBUNDLE中加载资源

UnityEngine.Objects can be loaded from AssetBundles using three distinct APIs that are all attached to the AssetBundleobject: LoadAsset, LoadAllAssets and LoadAssetWithSubAssets. All of these APIs also have asynchronous variants with the suffix -Async appended: LoadAssetAsync, LoadAllAssetsAsync, and LoadAssetWithSubAssetsAsync.

UnityEngine.Objects 可以从AssetBundle中通过3个不同的接口进行加载， 他们都属于AssetBundle对象： LoadAsset, LoadAllAssets 以及 LoadAssetWithSubAssets.所有者写接口都有带着Async后缀的异步方式：LoadAssetAsync, LoadAllAssetsAsync, 以及 LoadAssetWithSubAssetsAsync.

The synchronous API will always be faster than the asynchronous API by at least one frame. This particularly true in Unity 5.1 or older. Before Unity 5.2, all asynchronous APIs would load at most one Object per frame. This means thatLoadAllAssetsAsync and LoadAssetWithSubAssetAsync would be significantly slower than the corresponding synchronous API. This behavior was corrected in Unity 5.2. Asynchronous loads will now load multiple Objects per frame, up to their time-slice limits. See the [Low-level loading details](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#Lowlevel_Loading_Details) section for the underlying technical reasons for this behavior and for more details about time-slicing.

.同步的接口总是会比异步的接口快至少一帧。这在Unity5.1或者更老的版本中是完全正确的。在Unity5.2以前，所有异步接口每一帧只会加载最多一个对象。这意味着LoadAllAssetsAsync 以及LoadAssetWithSubAssetAsync 会比同样的同步接口慢上很多。这个行为在Unity5.2中被修复了。异步加载现在每一帧都会加载多个对象，取决于他们的时间片的限制。查看 [Low-level loading details](http://unity3d.com/cn/learn/tutorials/topics/best-practices/assetbundle-fundamentals?playlist=30089#Lowlevel_Loading_Details)部分了解这种行为的技术原因以及更多关于时间片的信息。

LoadAllAssets should be used when loading multiple independent UnityEngine.Objects. It should only be used when the majority (or all) of the Objects within an AssetBundle need to be loaded. Compared to the other two APIs, LoadAllAssets is slightly faster than multiple individual calls to LoadAssets. Therefore, if the number of assets to be loaded is large, and yet the number that needs to be loaded at a single time are less than 2/3rds of the total contents of the AssetBundle, consider splitting the AssetBundle into multiple smaller bundles and using LoadAllAssets.

LoadAllAssets 用于加载多个独立的UnityEngine.Objects。它只应该在AssetBundle主要的（或者全部）对象需要被加载时才使用。和另外两个接口相比LoadAllAssets 会比单独的执行多次LoadAssets略微快一些。所以，如果要加载的资源数量很大，并且需要一次加载的对象数量少于AssetBundle全部的2/3，那么考虑将AssetBundle拆分成多个小的包体然后使用LoadAllAssets。

LoadAssetWithSubAssets should be used when loading a composite Asset which contains multiple embedded Objects, such as an FBX model with embedded animations or a sprite atlas with multiple sprites embedded inside it. If the Objects that need to be loaded all come from the same Asset, but are stored in an AssetBundle with many other unrelated Objects, then use this API.

LoadAssetWithSubAssets  应该在加载一个包含了多个签入对象的复杂资源，例如FBX模型嵌入了动画或者sprite 图集包含了多个sprite。如果那些需要加载的对象都来源于一个Asset，但是和其他无关的对象都存储在一个AssetBundle中，那么就使用这个接口。

For any other case, use LoadAsset or LoadAssetAsync.

其他情况则使用LoadAsset 或者LoadAssetAsync。

### 3.5.1. Low-level loading details

底层加载细节

UnityEngine.Object loading is performed off the main thread: an Object's data is read from storage on a worker thread. Anything which does not touch thread-sensitive parts of the Unity system (scripting, graphics) will be converted on the worker thread. For example, VBOs will be created from meshes, textures will be decompressed, etc.

UnityEngine.Object 的加载是在主线程上进行的：一个Object的数据由工作线程从存储空间中读取。任何不涉及线程敏感部分的Unity系统（脚本，图形）都会被转移到工作线程中。例如通过网格，纹理生成的VBOs(顶点缓冲区对象)会被压缩等。

In versions prior to Unity 5.3, Object loading occurs in serial and certain portions of Object loading can only occur on the main thread. This is called "integration". After the worker thread finishes loading an Object's data, it pauses to integrate the newly-loaded Object onto the main thread, and remains paused (not loading the next Object) until main thread integration is complete.

在Unity5.3之前的版本中，对象的加载以串行的方式执行，并且其中的部分职能在主线程下执行。这被称为“一体化”。在工作线程完成了对象数据的加载之后，它停下来把最近加载的对象整合到主线程，并且保持暂停（而不是加载下一个对象）状态直到主线程的整合完成。

From Unity 5.3 onward, Object loading has been parallelized. Multiple Objects can be deserialized, processed and integrated on worker threads. When an Object finishes loading, its Awake callback will be invoked and the Object will become available to the rest of the Unity Engine during the next frame.

从Unity5.3开始，对象加载被并发化了。对个对象可以在工作线程上反序列化，处理并且整合。当一个对象完成加载时，它的Awake回调会被执行并且给对象会在下一帧对于其余的Unity引擎变成可用的。

The synchronous AssetBundle.Load methods will pause the main thread until Object loading is complete. In versions prior to 5.3, the asynchronous AssetBundle.LoadAsync methods will not pause the main thread until they need to integrate Objects onto the main thread. They will also time-slice Object loading so that Object integration does not occupy more than a certain number of milliseconds of frame time. The number of milliseconds is set by the propertyApplication.backgroundLoadingPriority:

同步的AssetBundle.Load 方法会暂停主线程直到对象的加载完成。在5.3以前的版本中。异步的AssetBundle.LoadAsync方法不会暂停主线程直到他们需要将对象整合到主线程时。他们也会讲对象的加载时间片化，所以对象整合在帧时间内不会占用超过一定数量的毫秒。毫秒的数量通过设置Application.backgroundLoadingPriority:进行控制：

* ThreadPriority.High: Maximum 50 milliseconds per frame

ThreadPriority.High:每一帧最多50毫秒

* ThreadPriority.Normal: Maximum 10 milliseconds per frame

ThreadPriority.Normal: 每一帧最多10毫秒

* ThreadPriority.BelowNormal: Maximum 4 milliseconds per frame

ThreadPriority.BelowNormal:每一帧最多4毫秒

* ThreadPriority.Low: Maximum 2 milliseconds per frame.

ThreadPriority.Low: 每一帧最多2毫秒

In Unity 5.1 and older, the asynchronous APIs will only integrate one Object per frame. This was considered a bug, and was fixed in Unity 5.2. From Unity 5.2 onwards, multiple Objects will be loaded until the frame-time limit for Object loading is reached. AssetBundle.LoadAsync will always take longer to complete than the comparable synchronous API (assuming all other factors are equal) because of the minimum one-frame delay between issuing the LoadAsync call and the object becoming available to the Engine.

在Unity5.1或者更老的版本中，异步的接口每一帧只会整合一个对象。这被认为是一个bug，并且在Unity5.2中已经修复。从Unity5.2开始对个对象会被加载直到达到帧时间的限制。AssetBundle.LoadAsync 相对于同步接口（假设所有的其他因素都是一样的）总是会使用更长的时间来完成，因为在调用LoadAsync 以及对象变成引擎可用的之间至少存在一帧的延迟。

Tests with real-world Objects and Assets demonstrate the difference. Prior to 5.2, loading a certain large texture on a low-end device would require 7ms synchronously or 70ms asynchronously. After 5.2, the observed difference was close to zero.

以现实世界的对象和资源进行测试以看出差异。在Unity5.2之前，在低端设备上加载一个非常大的纹理同步的需要7ms异步的需要70ms。在5.2之后，它们之前可以观察到的差异几乎为0。

### 3.5.2. AssetBundle dependencies

AssetBundle依赖

In Unity 5's AssetBundle system, the dependencies between AssetBundles are automatically tracked via two different APIs, depending on the runtime environment. In the Unity Editor, AssetBundle dependencies can be queried via the[AssetDatabase](http://docs.unity3d.com/ScriptReference/AssetDatabase.html) API. AssetBundle assignments and dependencies can be accessed and changed via the [AssetImporter](http://docs.unity3d.com/ScriptReference/AssetImporter.html) API. At runtime, Unity provides an optional API to load the dependency information generated during an AssetBundle build via a ScriptableObject-based [AssetBundleManifest](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.html) API.

在Unity5的AssetBundle系统中，AssetBundle之间的依赖是通过两个不同的接口进行自动追踪的，接口取决于运行时的环境。在Unity编辑器中，AssetBundle依赖可以通过[AssetDatabase](http://docs.unity3d.com/ScriptReference/AssetDatabase.html)接口进行查询。AssetBundle的归属和依赖可以通过[AssetImporter](http://docs.unity3d.com/ScriptReference/AssetImporter.html)接口访问和修改。在运行时，Unity提供了一个可选的接口来加载在通过基于脚本对象的 [AssetBundleManifest](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.html)接口构建AssetBundle时生成的依赖信息

An AssetBundle is "dependent" upon another AssetBundle when one or more of the parent AssetBundle's UnityEngine.Objects refers to one or more of the other AssetBundle's UnityEngine.Objects. For more information on inter-Object references, see the [Inter-Object references](http://unity3d.com/assets-objects-and-serialization#InterObject_References) section of the [Assets, Objects and Serialization](http://unity3d.com/assets-objects-and-serialization) article.

当一个或者更多的父AssetBundle的UnityEngine.Objects引用了一个或者更多的AssetBundle的UnityEngine.Objects时，我们称之为一个AssetBundle“依赖”于另一个AssetBundle。更多关于对象间引用的信息，参见 [Assets, Objects and Serialization](http://unity3d.com/assets-objects-and-serialization)文章的[Inter-Object references](http://unity3d.com/assets-objects-and-serialization#InterObject_References) 部分。

As described in the [Serialization and instances](http://unity3d.com/assets-objects-and-serialization#Serialization_and_Instances) section of that article, AssetBundles serve as sources for the source data identified by the FileGUID & LocalID of each Object contained within the AssetBundle.

像那篇文章中 [Serialization and instances](http://unity3d.com/assets-objects-and-serialization#Serialization_and_Instances)部分介绍的那样，AssetBundle们作为

AssetBundle内包含的每一个被FileGUI以及LocalID进行标示的对象的源数据的资源。

Because an Object is loaded when its Instance ID is first dereferenced, and because an Object is assigned a valid Instance ID when its AssetBundle is loaded, the order in which AssetBundles are loaded is not important. Instead, it is important to load all AssetBundles that contain dependencies of an Object before loading the Object itself. Unity will not attempt to automatically load any child AssetBundles when a parent AssetBundle is loaded.

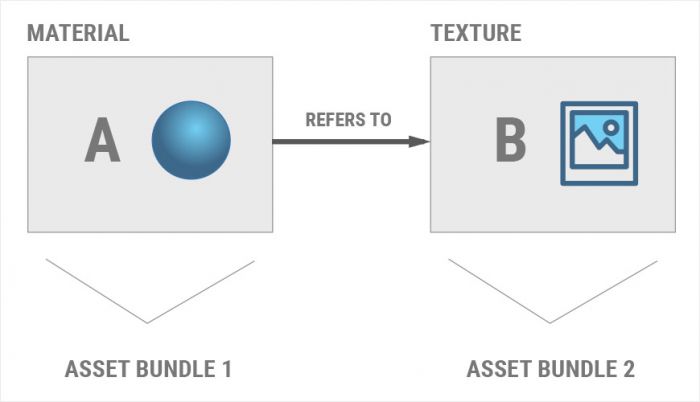
因为一个对象被加载时它的实例ID是首先被间接引用的，并且因为一个对象在它的AssetBundle被加载时才分配一个有效的实例ID， AssetBundle的加载顺序并不重要。相应的，在加载某个对象之前加载所有包含它所依赖的对象的AssetBundle是非常重要的。Unity不会在父AssetBundle被加载时自动尝试加载任何一个子AssetBundle。

**Example:**

**例子：**

Assume material A refers to texture B. Material A is packaged into AssetBundle 1, and texture B is packaged into AssetBundle 2.

假设材质A引用了纹理B。材质A被打包进了AssetBundle1,并且纹理B被打包进去了AssetBundle2。



In this use case, AssetBundle 2 must be loaded prior to loading Material A out of AssetBundle 1.

在这里例子中，AssetBundle2的加载必须先于材质A在AssetBundle1的加载

This does not imply that AssetBundle 2 must be loaded before AssetBundle 1, or that Texture B must be loaded explicitly from AssetBundle 2. It is sufficient to have AssetBundle 2 loaded prior to loading Material A out of AssetBundle 1.

折并不意味着AssetBundle2必须在AssetBundle1前加载，或者纹理B必须明确的在AssetBundle2中加载。只需要保证在AssetBundle1中加载材质A之前加载AssetBundle2。

Unity will not automatically load AssetBundle 2 when AssetBundle 1 is loaded. This must be done manually via a script. The specific AssetBundle APIs used to load AssetBundles 1 & 2 are irrelevant. AssetBundles loaded viaWWW.LoadFromCacheOrDownload can be freely mixed with AssetBundles loaded via AssetBundle.LoadFromFile orAssetBundle.LoadFromMemoryAsync.

Unity不会在AssetBundle1加载时自动加载AssetBundle2。这必须通过脚本手动完成。具体用于加载AssetBundle1和2的接口是无关紧要的。通过WWW.LoadFromCacheOrDownload 加载的AssetBundles可以和通过AssetBundle.LoadFromFile 或者AssetBundle.LoadFromMemoryAsync.混用。

### 3.5.3. AssetBundle manifests

When executing the AssetBundle build pipeline via the BuildPipeline.BuildAssetBundles API, Unity serializes an Object containing each AssetBundle's dependency information. This data is stored in a separate AssetBundle, which contains a single Object of the [AssetBundleManifest](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.html) type.

当通过BuildPipeline.BuildAssetBundles 接口执行AssetBundle的构建管道时，Unity将对象和每一个AssetBundle的依赖信息进行序列化。这些数据被存储在一个单独的AssetBundle中，其中包含了一个单独的[AssetBundleManifest](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.html) 类型对象。

This Asset will be stored in an AssetBundle with the same name as the parent directory where the AssetBundles are being built. If a project builds its AssetBundles to a folder at (projectroot)/build/Client/, then the AssetBundle containing the manifest will be saved as (projectroot)/build/Client/Client.manifest.

The AssetBundle containing the manifest can be loaded, cached and unloaded just like any other AssetBundle.

这个Asset会被存储在一个AssetBundle中，名字和AssetBundle构建的目录父节点名称一致。如果一个项目将AssetBundle构建在 (projectroot)/build/Client/中，那么AssetBundle包含了一个manifest会被存储为(projectroot)/build/Client/Client.manifest. 包含了manifest的AssetBundle是可以加载，缓存和卸载的，就和其他AssetBundle一样。

The AssetBundleManifest Object itself provides the [GetAllAssetBundles](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetAllAssetBundles.html) API to list all AssetBundles built concurrently with the manifest and two methods to query the dependencies of a specific AssetBundle:

AssetBundleManifest对象自身提供了 [GetAllAssetBundles](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetAllAssetBundles.html)接口来列出所有的AssetBundles同时建立的manifest以及两个方法来查询特定AssetBundle的依赖：

[AssetBundleManifest.GetAllDependencies](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetAllDependencies.html) returns all of an AssetBundle's dependencies, including the dependencies of the AssetBundle's direct children, its children's children, etc.

[AssetBundleManifest.GetAllDependencies](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetAllDependencies.html) 返回一个AssetBundle的所有依赖，包含了AssetBundle的子包以及子包的子包等。

[AssetBundleManifest.GetDirectDependencies](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetDirectDependencies.html) returns only an AssetBundle's direct children.

[AssetBundleManifest.GetDirectDependencies](http://docs.unity3d.com/ScriptReference/AssetBundleManifest.GetDirectDependencies.html) 只返回AssetBundle直接的子包。

Note that both of these APIs allocate arrays of strings. Use them sparingly, and preferably not during performance-sensitive portions of an application's lifetime.

注意所有这些接口都会分配字符串数组。请谨慎的使用它们，最好不要在应用程序性能敏感的阶段使用。

### 3.5.4. Recommendations

建议

It is considered a best practice to load as many needed Objects as possible before users enter performance-critical portions of an application, such as the main game level or world. This is particularly critical on mobile platforms, where access to local storage is slow and the memory churn of loading and unloading Objects at play-time can trigger the garbage collector.

最好的方式是在用户进入应用程序性能关键部分（例如游戏主关卡或者主世界）前，尽可能按需加载需要的对象。这在移动设备上尤其重要，因为对本地存储空间的访问很慢并且播放时加载卸载对象的内存损耗可以触发垃圾回收

For projects that must load and unload Objects while the application is interactive, see the [Managing loaded assets](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Managing_Loaded_Assets) section of the [AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns) article for more information on unloading Objects and AssetBundles.

对于那些必须要在应用程序交互式加载和卸载对象的项目来说，查看[AssetBundle usage patterns](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns)文章的[Managing loaded assets](http://unity3d.com/learn/tutorials/topics/best-practices/asset-bundle-usage-patterns#Managing_Loaded_Assets)部分来获取更多关于AssetBundle和卸载对象的信息。