**Asset Bundle Compression**

**资源包压缩**

Unity supports three compression options for Asset Bundles: LZMA, LZ4, and Uncompressed.

Unity支持3种AssetBundle的压缩方式:LZMA,LZ4以及不压缩

**LZMA Format**

**LZMA 格式**

By default, when Asset Bundles are built, they are stored in a compressed format. The standard compressed format is a single [LZMA](https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm) stream of serialized data files, and needs to be decompressed in its entirety before use.

当资源包生成时，他们默认会以压缩的方式存储。标准的压缩格式是序列化数据文件的单一[LZMA](https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Markov_chain_algorithm)流，并且在使用前需要整体解压缩

LZMA-Compressed bundles give the smallest possible download size, but has relatively slow decompression resulting in higher apparent load times.

进过LZMA压缩过的资源包会有最小的下载大小，但是在加载时的解压速度会相对较慢。

**LZ4 Format**

**LZ4格式**

Unity also supports [LZ4](http://cyan4973.github.io/lz4/) compression, which results in larger compressed file sizes, but does not require the entire bundle to be decompressed before use. LZ4 is a “chunk-based” algorithm, and therefore when objects are loaded from an LZ4-compressed bundle, only the corresponding chunks for that object are decompressed. This occurs on-the-fly, meaning there are no wait times for the entire bundle to be decompressed before use. The LZ4 Format was introduced in Unity 5.3 and was unavailable in prior versions.

Unity 同样支持[LZ4](http://cyan4973.github.io/lz4/)压缩格式，压缩后的文件大小会更大，但是不需要在使用前对整个包进行解压。LZ4是一种“基于块”的算法，所以从LZ4压缩的包中加载资源时，只有那些对象响应的块会被解压缩。这是即时发生的，意味着不需要为整个包的解压而进行等待。LZ4格式在Unity5.3中引入并且在之前的版本中不支持。

**Uncompressed Format**

**不压缩格式**

The third compression option is no compression at all. Uncompressed bundles are large, but are the fastest to access once downloaded.

第三种压缩选项就是完全不压缩。

不压缩的包会很大，但是下载完成后的访问速度是最快的。

**Caching of Compressed Bundles**

**对压缩后的包进行缓存**

The [WWW.LoadFromCacheOrDownload](http://docs.unity3d.com/540/Documentation/ScriptReference/WWW.LoadFromCacheOrDownload.html) function downloads and caches asset bundles to disk and thus greatly speeds up loading afterwards. From Unity 5.3 onwards, cached data can also be compressed with the LZ4 algorithm. This saves 40%–60% of space compared to uncompressed bundles. Recompression happens during download and thus is almost unnoticeable by the end users. As data arrives from the socket, Unity will decompress it and recompress it in LZ4 format. This recompression occurs during the download streaming, which means the cache compression begins as soon as enough of the data is downloaded, and continues incrementally until the download is complete. After that, data is read from the cached bundle by decompressing chunks on-the-fly when needed.

[WWW.LoadFromCacheOrDownload](http://docs.unity3d.com/540/Documentation/ScriptReference/WWW.LoadFromCacheOrDownload.html) 函数会下载并且将包缓存在磁盘中以便之后更快速的访问。自从Unity5.3开始，缓存的数据也是可以进行LZ4算法压缩的。相对于没有压缩的包，这会节省40%-60%的空间。再压缩会在下载的过程发生，所以用户最终几乎无法察觉。当数据通过socket传达时，Unity会解压他们并且重新以LZ4格式进行压缩。这里的再压缩在下载数据流的时候进行。这意味着缓存压缩会在足够的数据下载完后就开始，并且持续进行直至整个下载完成。在这之后，需要访问数据时会通过解压缩缓存包中的块进行即时访问。

Cache compression is enabled by default and is controlled by the [Caching.compressionEnabled](http://docs.unity3d.com/540/Documentation/ScriptReference/Caching-compressionEnabled.html) property. It affects bundles cached to disk and stored in memory.

缓存压缩默认是开启的，可以通过[Caching.compressionEnabled](http://docs.unity3d.com/540/Documentation/ScriptReference/Caching-compressionEnabled.html)属性进行控制。他会导致资源包被缓存在磁盘上并且存储在内存中。

**AssetBundle load API overview**

**资源包加载API概略**

This table provides a comparison of memory and performance overheads when using different compression types and different loading methods.

这个表格提供了使用不同的压缩格式和不同加载方式时的内存和性能开销对照。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Uncompressed** | **Chunk Compressed (LZ4)** | **Stream Compressed (LZMA)** |
| **WWW \*** | Memory: uncompressed bundle size + (while WWW is not disposed, uncompressed bundle size). Performance: no extra processing. | Memory: LZ4HC compressed bundle size + (while WWW is not disposed, LZ4HC compressed bundle size). Performance: no extra processing. | Memory: LZ4 compressed bundle size + (while WWW is not disposed, LZMA compressed bundle size). Performance: LZMA decompression + LZ4 compression during download. |
| **LoadFromCacheOrDownload** | Mem: no extra memory is used. Perf: reading from disk. | Mem: no extra memory is used. Perf: reading from disk. | Mem: no extra memory is used. Perf: reading from disk. |
| **LoadFromMemory (Async)** | Mem: uncompressed bundle size. Perf: no extra processing. | Mem: LZ4HC compressed bundle size. Perf: no extra processing. | Mem: LZ4 compressed bundle size. Perf: LZMA decompression + LZ4 compression. |
| **LoadFromFile(Async)** | Mem: no extra memory is used. Perf: reading from disk. | Mem: no extra memory is used. Perf: reading from disk. | Mem: LZ4 compressed bundle size. Perf: reading from disk + LZMA decompression + LZ4 compression. |
| **WebRequest (also supports caching)** | Mem: uncompressed bundle size. Perf: no extra processing [+reading from disk if cached]. | Mem: LZ4HC compressed bundle size. Perf: no extra processing [+reading from disk if cached]. | Mem: LZ4 compressed bundle size. Perf: LZMA decompression + LZ4 compression during download [+reading from disk if cached]. |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **不压缩** | **块压缩 (LZ4)** | **流压缩 (LZMA)** |
| **WWW \*** | 内存: 未压缩的资源包大小 + (WWW没有关闭时, 未压缩的资源包大小). 性能: 没有额外的处理. | 内存: LZ4HC格式压缩的资源包大小+ (WWW没有关闭时, LZ4HC格式压缩的资源包大小). 性能: 没有额外的处理. | 内存: LZ4格式压缩的资源包大小 + (WWW没有关闭时, LZMA格式压缩的资源包大小). 性能: LZMA 解压缩 + 下载时LZ4 压缩. |
| **LoadFromCacheOrDownload** | 内存: 没有额外的内存开销. 性能: 从磁盘读取. | 内存: 没有额外的内存开销. 性能: 从磁盘读取. | 内存: 没有的内存开销. 性能: 从磁盘读取. |
| **LoadFromMemory (Async)** | 内存: 没有压缩的包大小. 性能: 没有额外的处理. | 内存: LZ4HC格式压缩的包大小. 性能: 没有额外的处理. | 内存: LZ4格式压缩的包大小. 性能: LZMA 解压缩 + LZ4 压缩. |
| **LoadFromFile(Async)** | 内存: 没有额外的内存开销. 性能: 从磁盘读取. | 内存: 没有额外的内存开销. 性能: 从磁盘读取. | 内存: LZ4格式压缩的包大小. 性能: 从磁盘读取+ LZMA 解压缩 + LZ4 压缩. |
| **WebRequest (also supports caching)** | 内存: 没有压缩的包大小. 性能: 没有额外的处理[+从磁盘读取如果进行了缓存的话]. | 内存: LZ4HC 格式压缩的包大小. 性能: 没有额外的处理[+从磁盘读取如果进行了缓存的话]. | 内存: LZ4 compressed bundle size. 性能: LZMA 解压缩 + 下载时的LZ4 压缩 [+从磁盘读取如果进行了缓存的话]. |

\* *When downloading a bundle using WWW, WebRequest there is also an 8x64KB accumulator buffer which stores data from a socket.*

\* 当你通过WWW,WebRequest进行资源包的下载时，还会有一个8x64k大小的累加器缓冲区来保存从socket传入的数据。

Thus, use the following guidelines when using low-level loading API in your games:

所以，在游戏中使用底层的API时参考以下的指导：

1. Deploying asset bundles with your game as StreamingAssets - use BuildAssetBundleOptions.ChunkBasedCompression when building bundles and AssetBundle.LoadFromFileAsync to load it. This gives you data compression and the fastest possible loading performance with a memory overhead equal to read buffers.

1. 以StreamingAssets的方式发布游戏的资源包，使用BuildAssetBundleOptions.ChunkBasedCompression来生成资源包并且使用AssetBundle.LoadFromFileAsync来加载他。这会提供给你数据的压缩以及最快的加载速度，并且内存开销和读取缓冲区的大小一样。

2. Downloading asset bundles as DLCs - use default build options (LZMA compression) and LoadFromCacheOrDownload/WebRequest to download and cache it. Here you’ll have the best possible compression ratio and AssetBundle.LoadFromFile loading performance for further loads.

2. 下载资源包作为DLCs（可下载资料包）- 使用默认的生成选项（LZMA压缩）以及LoadFromCacheOrDownload/WebRequest 来下载并且缓存他们。这样你会拥有最好的压缩比，但是使用AssetBundle.LoadFromFile进行加载会有额外的开销。

3. Encrypted bundles - choose BuildAssetBundleOptions.ChunkBasedCompression and use LoadFromMemoryAsync for loading (this is pretty much the only scenario where LoadFromMemory[Async] should be used).

3. 加密资源包-使用BuildAssetBundleOptions.ChunkBasedCompression（生成）以及LoadFromMemoryAsync进行加载（这几乎是LoadFromMemory[异步]唯一应该被用到的地方）

4. Custom compression - use BuildAssetBundleOptions.UncompressedAssetBundle to build and AssetBundle.LoadFromFileAsync to load a bundle after it was decompressed by your custom compression algorithm.

4. 自定义压缩-使用BuildAssetBundleOptions.UncompressedAssetBundle生成并且使用你自定义的压缩算法来压缩他，再解压之后通过AssetBundle.LoadFromFileAsync来加载资源包。

You should generally always choose asynchronous functions - they don’t stall the main thread and they allow loading operations to be queued more efficiently. And absolutely avoid calling synchronous and asynchronous functions at the same time - this might introduce hiccups on the main thread.

通常情况下你应该总是选择异步的函数-他们不会阻塞主线程并且他们允许加载的操作被更有效的调度。并且绝对不要同时去执行同步和异步的方法 - 这会使得主线程有所卡顿。

**Compatibility**

**兼容性**

The Asset Bundle container format was changed in order to support new compression type, and to provide basis for further improvements. Unity 5 still supports bundles created in Unity 4, however bundles created in earlier version (2.x, 3.x) are not supported.

资源包容器格式为了支持新的压缩类型以及提供进一步优化的基础，做了改动。Unity5依然支持由Unity4生成的资源包，但是更早版本（2.x,3.x）是不支持的。