# Unity资源打包之Asset Bundle

_凉笙 (/u/9131c2f30f1b) 关注
2017.10.31 18:19* 字数 2654 阅读 5030 评论 2 喜欢 10
(/u/9131c2f30f1b)

***Asset Bundle*的作用：**

1.AssetBundle是一个压缩包包含模型、贴图、预制体、声音、甚至整个场景，可以在游戏运行的时候被加载；

2.AssetBundle自身保存着互相的依赖关系；

3.压缩包可以使用LZMA和LZ4压缩算法，减少包大小，更快的进行网络传输；

4.把一些可以下载内容放在AssetBundle里面，可以减少安装包的大小；

**什么是*AssetBundle***

可以归为两点：

1，它是一个存在于硬盘上的文件。可以称之为压缩包。这个压缩包可以认为是一个文件夹，里面包含了多个文件。这些文件可以分为两类：serialized file 和 resource files。（序列化文件和源文件）

serialized file：资源被打碎放在一个对象中，最后统一被写进一个单独的文件（只有一个）

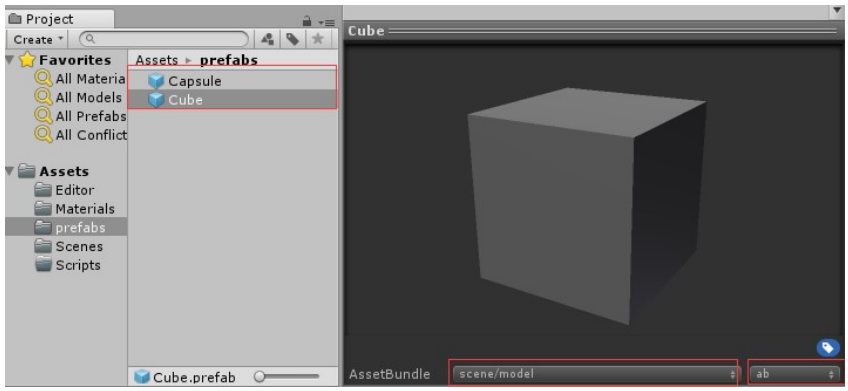
resource files：某些二进制资源（图片、声音）被单独保存，方便快速加载

2，它是一个AssetBundle对象，我们可以通过代码从一个特定的压缩包加载出来的对象。这个对象包含了所有我们当初添加到这个压缩包里面的内容，我们可以通过这个对象加载出来使用。

## Asset Bundle资源打包实例

无论是模型资源还是UI资源，最好是先把他们放在Prefab中，然后在做成Assetbundle。我们以模型来举例，Assetbundle中可以放一个模型、也可以放多个模型，它是非常灵活了那么最需要考虑的就是模型空间占用的问题。

下面我们来实际操作下，首先随便先创建两个3D对象，Cube和Capsule,并将他们做成Prefab,然后去指定资源的AssetBundle属性，这里我将这两个模型都打包成model.ab包（xxxa/xxx）这里xxxa会生成目录，名字为xxx ，后面的ab是后缀名，可自己制定。




Paste_Image.png

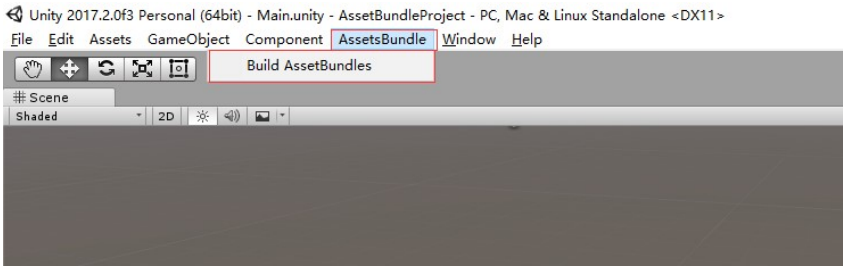
设置好属性后，下面开始构建AssetBundle包，首先先创建一个文件夹命名Editor，这个文件夹是不会进行打包的特定编辑器扩展文件夹。然后我们创建一个编辑器扩展类CreateAssetbundles。写入下面的代码

```
using UnityEditor;
using System.IO;

public class CreateAssetbundles  {

    [MenuItem("AssetsBundle/Build AssetBundles")]
     static void BuildAllAssetBundles()//进行打包
    {
        string dir = "AssetBundles";
        //判断该目录是否存在
        if (Directory.Exists(dir) == false)
        {
            Directory.CreateDirectory(dir);//在工程下创建AssetBundles目录
        }
        //参数一为打包到哪个路径，参数二压缩选项   参数三 平台的目标
        BuildPipeline.BuildAssetBundles(dir, BuildAssetBundleOptions.None,BuildTarge
    }
}
```



BuildAssetBundleOptions.None：使用LZMA算法压缩，压缩的包更小，但是加载时间更长。使用之前需要整体解压。一旦被解压，这个包会使用LZ4重新压缩。使用资源的时候不需要整体解压。在下载的时候可以使用LZMA算法，一旦它被下载了之后，它会使用LZ4算法保存到本地上。
BuildAssetBundleOptions.UncompressedAssetBundle：不压缩，包大，加载快
BuildAssetBundleOptions.ChunkBasedCompression：使用LZ4压缩，压缩率没有LZMA高，但是我们可以加载指定资源而不用解压全部。
注意使用LZ4压缩，可以获得可以跟不压缩想媲美的加载速度，而且比不压缩文件要小。

然后回到Unity里面点击我们刚刚扩展出来的打包按钮




Paste_Image.png

点击后我们的模型就打包了出来，可以在工程的目录下可以找到AssetBundles目录，在AssetBundles下有个Scene文件夹里面就是我们的打包文件了，后缀是.ab


Paste_Image.png

## AssetBundle的加载

AssetBundle的加载有以下几种方式，从内存加载使用LoadFromMemoryAsync，从本地文件加载可以使用LoadFromFile，从服务器上Web上加载可以使用UnityWbRequest。下面我们来看看这几种加载的方式。
首先可以先把我们Unity里面的两个模型的Prefab Cube和Capsule删除了,然后创建一个脚本挂在Camera上，打开脚本
**第一种加载方式*(LoadFromMemoryAsync)*从内存加载**

```
using UnityEngine;
using System.IO;
using System.Collections;
public class LoadFromFileExample : MonoBehaviour {

    IEnumerator  Start () {
        string path = "AssetBundles/scene/model.ab";
        //第一种加载AB的方式 LoadFromMemoryAsync
        //异步加载
        AssetBundleCreateRequest request = AssetBundle.LoadFromMemoryAsync(File.Read
        yield return request;
        AssetBundle ab = request.assetBundle;
        //同步方式
        //AssetBundle ab=  AssetBundle.LoadFromMemory(File.ReadAllBytes(path));

         //使用里面的资源
        Object[] obj = ab.LoadAllAssets<GameObject>();//加载出来放入数组中
        // 创建出来
        foreach (Object o in obj)
        {
            Instantiate(o);
        }
    }
}
```

**第二种方式*(LoadFromFile)*从本地加载**

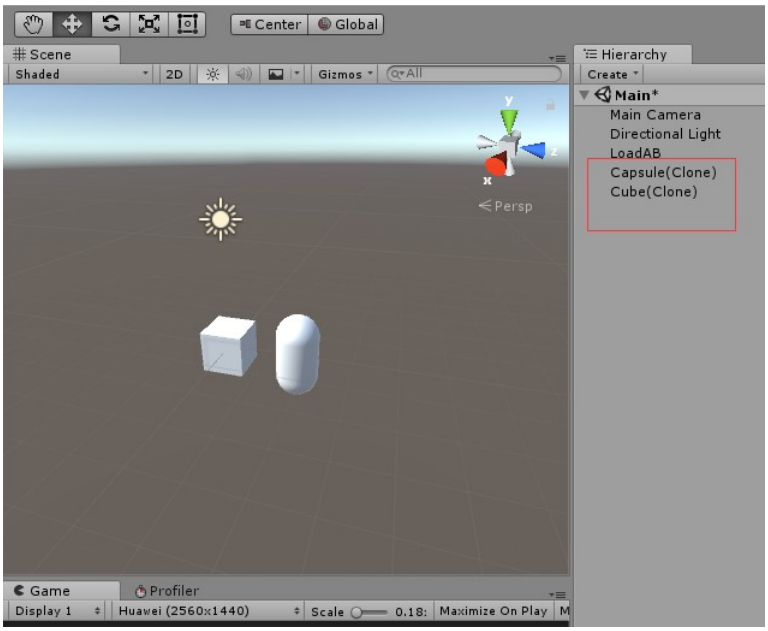
```
using UnityEngine;
using System.Collections;

public class LoadFromFileExample : MonoBehaviour {

    IEnumerator  Start () {
        string path = "AssetBundles/scene/model.ab";
        //第二种加载方式 LoadFromFile
        //异步加载
        AssetBundleCreateRequest request = AssetBundle.LoadFromFileAsync(path);
        yield return request;
        AssetBundle ab = request.assetBundle;
        //同步加载
        //AssetBundle ab = AssetBundle.LoadFromFile(path);

        //使用里面的资源
      Object[] obj = ab.LoadAllAssets<GameObject>();//加载出来放入数组中
        // 创建出来
        foreach (Object o in obj)
        {
            Instantiate(o);
        }
     }
}
```

**第三种方式*(UnityWbRequest)*从服务器或者本地加载**

```
using UnityEngine;
using System.Collections;
using UnityEngine.Networking;

public class LoadFromFileExample : MonoBehaviour {

    IEnumerator  Start () {
        //第三种加载方式    使用UnityWbRequest  服务器加载使用http本地加载使用file
        //string uri = @"file:///C:\Users\Administrator\Desktop\AssetBundleProject\A
        string uri = @"http://localhost/AssetBundles\model.ab";
        UnityWebRequest request = UnityWebRequest.GetAssetBundle(uri);
        yield return request.Send();
        AssetBundle ab = DownloadHandlerAssetBundle.GetContent(request);

        //使用里面的资源
      Object[] obj = ab.LoadAllAssets<GameObject>();//加载出来放入数组中
        // 创建出来
        foreach (Object o in obj)
        {
            Instantiate(o);
        }
    }
}
```



这样我们Model包里面的资源就加载出来并创建在场景里了。这时候运行Unity就可以看到两个模型都各自创建了出来




Paste_Image.png

当然也可以创建指定的资源例如

```
AssetBundle ab=AssetBundle.LoadFromFile("AssetBundles/scene/model.ab");
  GameObject go = ab.LoadAsset<GameObject>("Cube");
  Instantiate(go)
```

这样就实现了Asset Bundle资源的加载了

## AssetBundle分组策略

1，把经常更新的资源放在一个单独的包里面，跟不经常更新的包分离
2，把需要同时加载的资源放在一个包里面
3，可以把其他包共享的资源放在一个单独的包里面

4，把一些需要同时加载的小资源打包成一个包
5，如果对于一个同一个资源有两个版本，可以考虑通过后缀来区分 v1 v2 v3 unity3dv1 unity3dv2

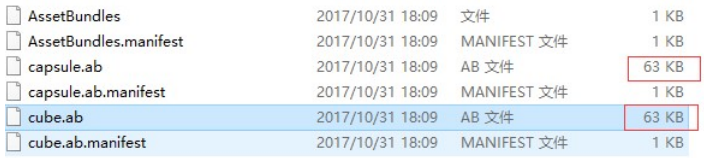


***1*，逻辑实体分组**
a,一个UI界面或者所有UI界面一个包（这个界面里面的贴图和布局信息一个包）
b,一个角色或者所有角色一个包（这个角色里面的模型和动画一个包）
c,所有的场景所共享的部分一个包（包括贴图和模型）
***2*，按照类型分组**
所有声音资源打成一个包，所有shader打成一个包，所有模型打成一个包，所有材质打成一个包
***3*，按照使用分组**
把在某一时间内使用的所有资源打成一个包。可以按照关卡分，一个关卡所需要的所有资源包括角色、贴图、声音等打成一个包。也可以按照场景分，一个场景所需要的资源一个包

## 依赖打包

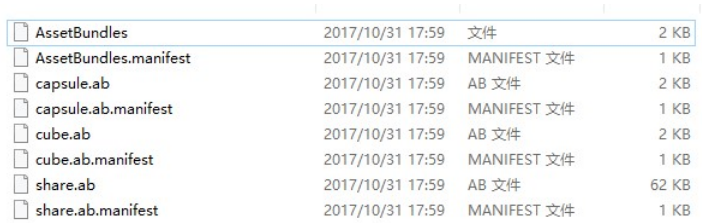


意思就是例如有两个模型使用的都是同一个材质和贴图，那么模型和材质贴图之间就是依赖关系。如果我们这两个模型都单独打包出来那么就会打包出两份材质和贴图，这样包就会变大，那么我们如何解决呢，这里Unity里面自带有一种方式，那就是首先先把所依赖的材质和贴图单独打包到一个文件夹中，然后再分别打包两个需要依赖这个材质和贴图的模型。这样Unity就会去查找这个材质贴图，发现这个材质和贴图已经打包了出来，那么它就不会去重复的打包材质和贴图了，这样就大大减小了包的大小




Paste_Image.png

上面一个是直接两个模型分别打包出来可以看到材质和贴图都分别打包了出来，分别都是63KB,而下面的是先将材质和贴图打包出来是62KB再将两个2KB的模型打包出来，总共也才64KB。




Paste_Image.png

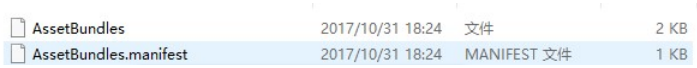
这就是Unity自带的依赖打包.
但是加载的时候模型、材质和贴图都要进行去加载，不然就会出现财政的丢失

```
using UnityEngine;

public class LoadFromFileExample : MonoBehaviour {

    void Start () {
        AssetBundle ab = AssetBundle.LoadFromFile("AssetBundles/cube.ab");
        AssetBundle abShare = AssetBundle.LoadFromFile("AssetBundles/share.ab");
        //GameObject go = ab.LoadAsset<GameObject>("Cube");
        //Instantiate(go);
        Object[] obj = ab.LoadAllAssets<GameObject>();//加载出来放入数组中
        //创建出来
        foreach (Object o in obj)
        {
            Instantiate(o);
        }
    }
}
```



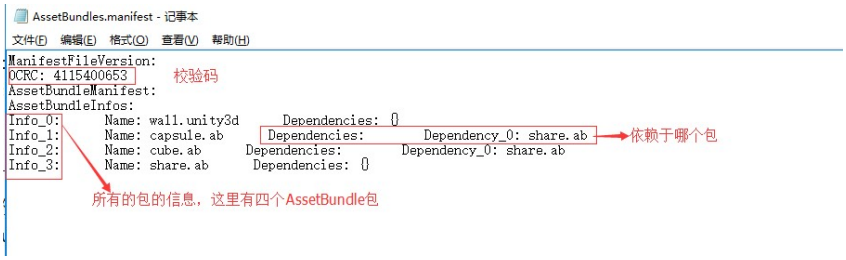
**使用*AssetBundleManifest*获取所有的包**

在打包AssetBundle后出现一个AssetBundles和一个AssetBundles.manifest两个文件，打包出来的所有的AssetBundle包都会放在AssetBundles里面。




Paste_Image.png

而AssetBundles.manifest是一个文本文件，里面有一些包的信息，下面可以打开看看



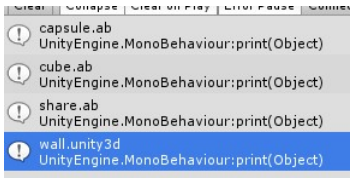

Paste_Image.png

所以下面我们就可以去读取到AssetBundles然后获取里面所有的AssetBundle包

在Start方法里面写入

```
    AssetBundle manifesAB = AssetBundle.LoadFromFile("AssetBundles/AssetBundles");
       AssetBundleManifest manifest= manifesAB.LoadAsset<AssetBundleManifest>("Asse
       foreach (string name in manifest.GetAllAssetBundles())
       {
           print(name);
       }
```

这时候运行Unity就可以看到所有包都完整的输出出来了。

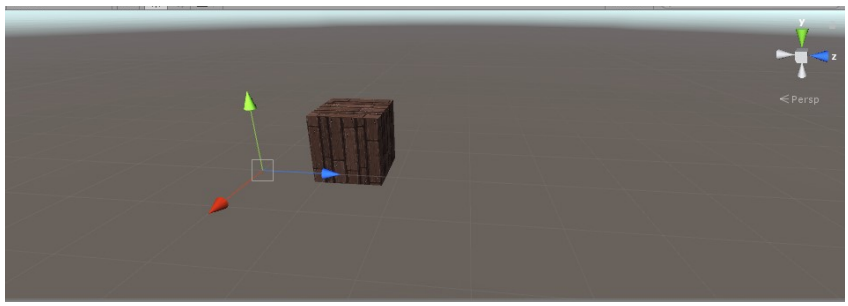



Paste_Image.png

**利用*Manifest*加载某个包所依赖的包**

```
AssetBundle manifesAB = AssetBundle.LoadFromFile("AssetBundles/AssetBundles"
AssetBundleManifest manifest= manifesAB.LoadAsset<AssetBundleManifest>("Asse
foreach (string name in manifest.GetAllAssetBundles())
{
    print(name);
}
string []strs=manifest.GetAllDependencies("Cube.ab");
foreach (var name in strs)
{
    AssetBundle.LoadFromFile("AssetBundles/"+name);
}
```

运行后可以看到所依赖的Share包的资源也加载了出来




Paste_Image.png

## AssetBundle的卸载

卸载有两个方面
1，减少内存使用
2，有可能导致丢失
所以什么时候去卸载资源
AssetBundle.Unload(true)卸载所有资源，即使有资源被使用着
（1，在关卡切换、场景切换的时候
（2，资源没被调用的时候
AssetBundle.Unload(false)卸载所有没用被使用的资源
个别资源怎么卸载
(1，通过 Resources.UnloadUnusedAssets.
(2，场景切换的时候

## 文件校验

文件校验可以在文件传输的时候保证文件的完整性，例如A在给我传输了一个文件之前会生成一个校验码，对于这个文件只会生成这一个唯一的校验码，只要传输给我的文件有一点不一样那么校验码就会完全不同。所以A在传输给我文件的时候会把文件和校验码都传输给我，当我取到这个文件的时候我也会使用和A同样一个算法去生成这个文件的校验码，然后拿这个值和A传输给我的校验码比对，如果一样说明这个文件是完整的，如果不一样那么就重新传输。下面是几个算法生成的校验值
CRC MD5 SHA1
相同点：
CRC、MD5、SHA1都是通过对数据进行计算，来生成一个校验值，该校验值用来校验数据的完整性。
不同点：

1. 算法不同。CRC采用多项式除法，MD5和SHA1使用的是替换、轮转等方法；
2. 校验值的长度不同。CRC校验位的长度跟其多项式有关系，一般为16位或32位；MD5是16个字节（128位）；SHA1是20个字节（160位）；
3. 校验值的称呼不同。CRC一般叫做CRC值；MD5和SHA1一般叫做哈希值（Hash）或散列值；

4. 安全性不同。这里的安全性是指检错的能力，即数据的错误能通过校验位检测出来。CRC的安全性跟多项式有很大关系，相对于MD5和SHA1要弱很多；MD5的安全性很高，不过大概在04年的时候被山东大学的王小云破解了；SHA1的安全性最高。
5. 效率不同，CRC的计算效率很高；MD5和SHA1比较慢。
6. 用途不同。CRC一般用作通信数据的校验；MD5和SHA1用于安全（Security）领域，比如文件校验、数字签名等。

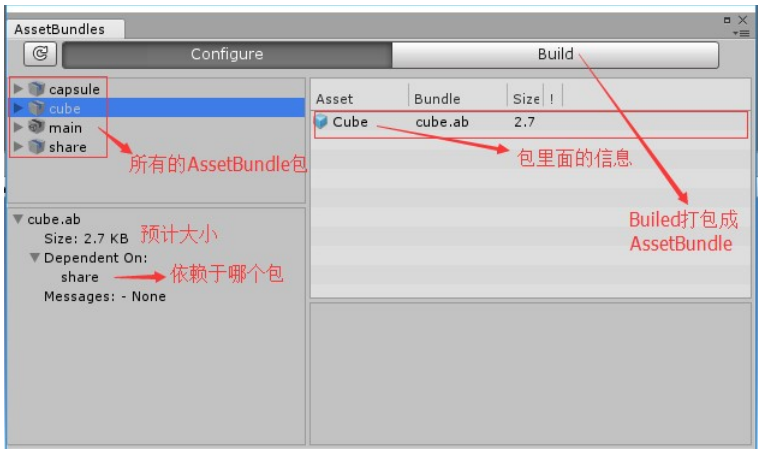


### Unity Asset Bundle Browser tool

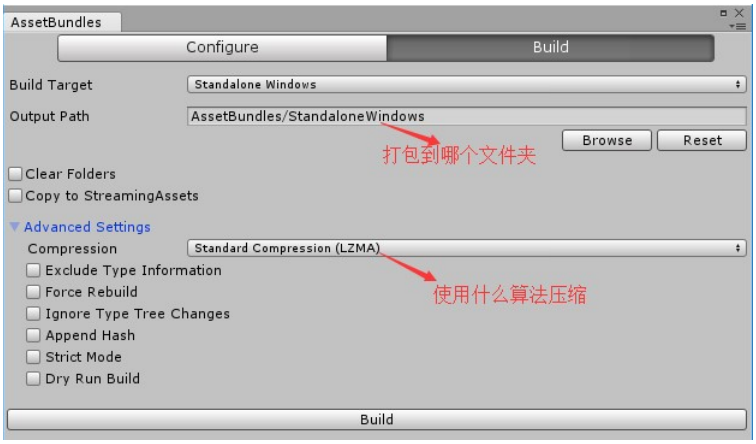
这是一个AssetBundle的查看工具，是Unity官方发布的一个扩展工具，可以查看帮助打包AssetBundle和查看AssetBundle内容。可以去GitHub上下载 https://github.com/Unity-Technologies/AssetBundles-Browser (https://link.jianshu.com?t=https://github.com/Unity-Technologies/AssetBundles-Browser)
下载后直接将里面的Editor扩展工具拖入我们的Unity Project工程中


Paste_Image.png

然后再窗口Window下找到并选择AssetBundle Browser选项，就可以打开看到我们Asset Bundle 窗口了




Paste_Image.png




Paste_Image.png

这是一个轻量级的AssetsBundle使用工具，里面可以打包可以查看打包的内容可以删除打包的内容，非常好用