# AssetBundle and the AssetBundle Manager

## 介绍

AssetBundle允许从本地或者远程服务器加载Assets资源，利用AssetBundles技术，Assets资源可以放在远程服务器上，这种技术增加了项目灵活性并且减少项目初始包的大小。

本文介绍AssetBundles并且讨论一步一步的介绍怎么样使用它，怎样将资源打包到AssetBundle中，如何使用以及如何处理资源的引用计数，所有这些我们都可以使用AssetBundle Manader来简化AssetBundle构建、测试和发布环节。最后会介绍一个使用AB和Variants实际项目的例子。

## 示例

在读本文开始之前，最好先从官方下载下载[示例](https://gitee.com/AllanUnity/AssetBundleManager)

## 什么是AssetBundle

AssetBundles是Unity编辑器中在编辑过程中创建的一些文件，这些文件可以在项目运行环境中使用。AssetBundles可以包含的资源文件比如模型，材质，贴图和场景等等，注意：AssetBundles不能包含脚本!

具体来说，一个AssetBundle就是把资源或者场景以某种方式紧密集合在一起的一个文件。这个AssetBundle文件可以被单独加载到unity应用程序中。这允许模型、贴图、音效设置很大的场景这些资源进行流式加载或者异步加载。当然AssetBundle文件也可以本地缓存这样能够在程序启动的时候立马被加载出来。但AssetBundle技术主要目的就是需要的时候从远程服务器下载需要的资源。AssetBundle可以包含任何unity可识别的资源文件，甚至包括二进制文件。但就是不能包含脚本文件。

有很多AssetBundle的使用案例。资源在程序中可以被加载和释放。Post-release DLC可以被轻易的实现。应用程序发布的时候可以使得包体变得更小，在应用程序启动的时候加载需要加载的资源。应用程序国际化也会变得相对简单，我们可以根据玩家的地理位置来判断加载对应的资源文件。应用程序可以用新的资源文件来达到修复、更新的目的。

AssetBundles怎么管理和规划更取决于具体的项目需求。以下有一些基本的规则以便我们更好的理解AssetBundle。

* AssetBundle需要整体的下载和缓存的
* AssetBundle不需要整体的加载到应用程序中
* Assets在AssetBundles中是具有相互依赖性的
* Assets在AssetBundle中可以和其他资源共享依赖
* 每一个AssetBundle都有一些技术开销，在文件加载上和对文件的管理上
* AssetBundle必须针对平台进行打包

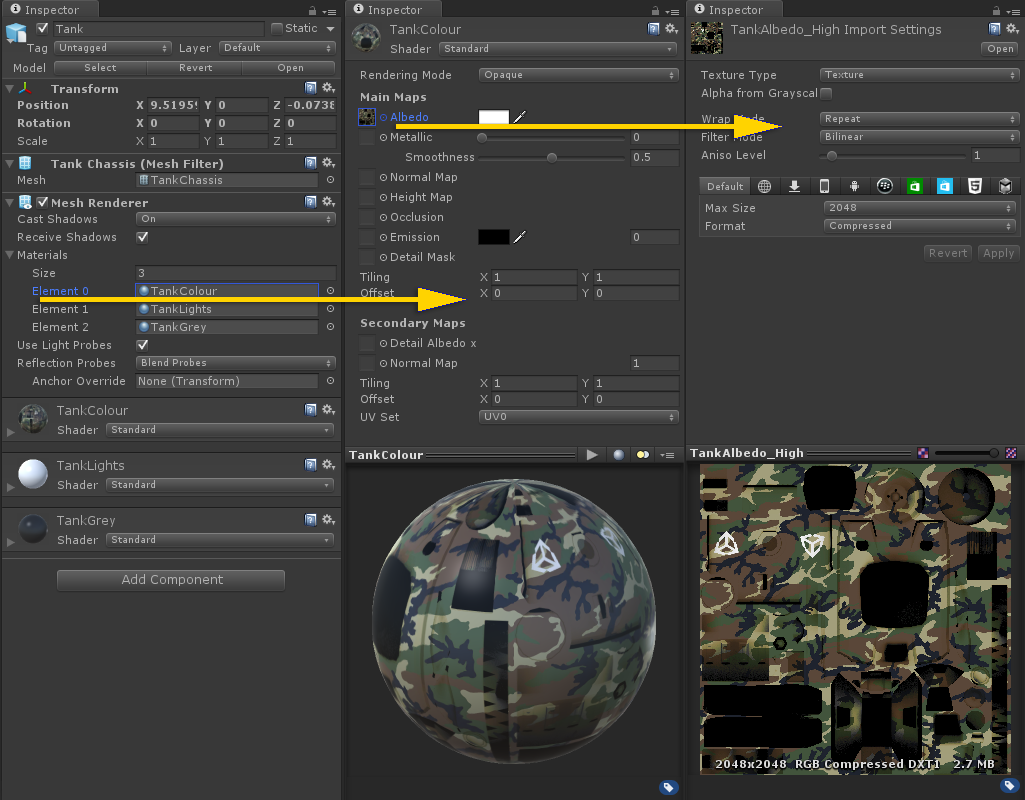
每一个AssetBundle都是被整体下载的，就算它不会立马被用到甚至不会在当前场景中用到，它也会占用下载和磁盘空间资源。

一个AB文件被整体下载之后，我们可以根据需要去加载里面的资源。

一个资源可能会对其他资源有依赖关系，比如一个模型可以有很多依赖关系，一个游戏中的模型不仅仅只有网格数据，实际上他是一个拥有很多依赖资源组成的一个共同体。

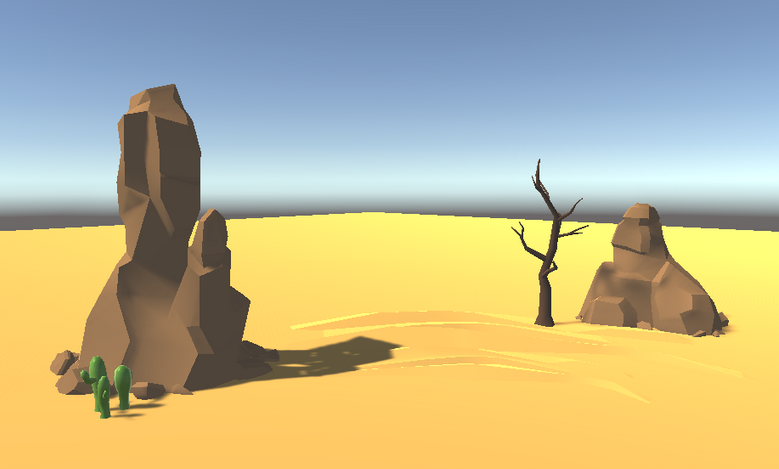


这个坦克模型的Mesh Render依赖材质资源，材质又依赖贴图资源。因此这个坦克模型依赖三个文件资源并不是仅仅就一个网格资源



这个坦克模型的资源依赖链：模型 > 材质 >纹理

Assets资源之间也是可以相互依赖的，例如：两个不同的模型可以共享相同的材质，这个材质是依赖贴图资源。



不同的岩石模型共享相同的材质

在组织AssetBundle结构的时候，需要去平衡开销，把AssetBundle分成若干个很小的AssetBundle的时候会很耗性能，但如果做成一个大的AssetBundle又会包含许多不需要用到的资源，所以要根据你实际的项目来平衡。

AssestBundle的目录被以优化的方式编译到目标平台上根据目标平台的导出设置，因为这样所以AssestBundles需要导出到各个目标平台。

## 依赖关系和依赖管理

关于依赖和依赖管理有几个重要的点

资源依赖永远不会丢失。如果这个依赖的assest在AssestBundle被创建时没有指定任何的AssestBundle，依赖的Assest连同选中的Assest将被添加到AssestBundle，这是非常方便和避免依赖Assest的损失。但是这也可能导致Assets重复。例如,使用两个岩石上面共享相同的Material,如果岩石列在单独的AssetBundles和Material没有显式地指定一个AssetBundle,将被添加到包含那种Material的两个 AssetBundles包中。值得注意的事，当这么做的时候，两个重复的资源将被存储在各自的AssetBundles而且依赖关系也破裂了，每个模型的Assest将依赖自己复制出来的Material，没有了共享Assest的任何优势。为了防止这种情况的发生,这些材料需要显式地指定一个AssetBundle，这可以实现对自己和其他资源的分享。用这种方法，这两个岩石的AssestBundle将依赖这个岩石Material。

关于依赖信息都存储在Mamnifest文件中，这个文件很像AssetBundle的资源的table文件。当AB构建时，unity统一生成大量数据，这些数据保存清单的细节，每一个目标平台都有一个对应的清单，清单列出所有为当前平台所需要的AssetBundles存储，跟踪和依赖项。使用清单可以查询所有AssetBundle和他们的依赖项。

有一个关于AssetBundles的特别的设置被叫做 AssetBundle Variants，AssetBundle Variants目的在于一种特殊的情况：在项目中重新映射一个不同Assest中的单个对象，这对于一个需要基于标准分辨率、语言、定位、或用户偏好选择不同的Assest特别有用。 AssetBundle Variants可以容纳所需的各种Asset的覆盖所有支持选择对象和所需的Asset可以根据需要被映射到该对象的选择的AssetBundle Variants。

AssetBundles文件包含资产文件,如模型、材料、材质和场景。AssetBundles在辑器编辑创建在游戏中使用，AssetBundles旨在从本地或远程加载资源。AssetBundles可以变异映射到对象根据现场用户的偏好。

# Working with AssetBundles and the AssetBundle Manager

## 介绍

对于使用AssetBundles的一个关键和重要的工作是对Asset构建测试，通常，AssetBundles会定期的发生改变，所以需要定期的创建AssetBundles，然后将其上传到一个远程服务器并测试托管的AssetBundles资源。本章重点介绍使用AssetBundles Manader。AssetBundle Manager提供了一个高级的API大大提高了工作流。

## 使用AssetBundles

使用AssetBundles有以下几步：

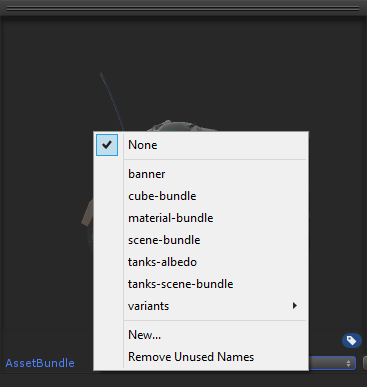
* 在编辑器里面构建AssetBundles
* 上传AssetBundles到外部存储
* 在运行时下载AssetBundles
* 从AssetBundles中加载物体 值得注意的是一些AssetBundles可以存储在本地，以保证立即加载，这有助于防止一个应用程序的安装不能从远程下载所有的AB资源。例如，当应用程序没有访问下载内容时将从本地AssetBundles加载默认语言和本地数据。

值得注意的是，一个AssetBundles会根据平台进行打包，AssetBundles根据导入的设置和目标平台中的设置构建为目标平台的AB资源。

在下面的场景中，一种方式就是打包包括地面，沙丘，岩石，仙人掌，树。这些场景允许包括所有独立的材质。坦克模型会有自己的AssetBundles，这样允许改变或者更新玩家信息。想要完成这个坦克的GameObject创建，需要额外依赖两个AssetBundle一个是材质球，一个是纹理，这么做对于更新这个材质球和纹理来说，将会造成最小的麻烦，也允许选择版本或者多个版本，从多个版本中选择一个需要的AssetBundle来实现平台，国际化或者目标设备的区分。 

在Editor模式下组织和构建AssetBundles，资源必须分配给一个AssetBundle，当查看一个资源，在Inspector面板中右下角可以看到这个AssetBundle的名字和Variant。预览窗口打开才能看到。

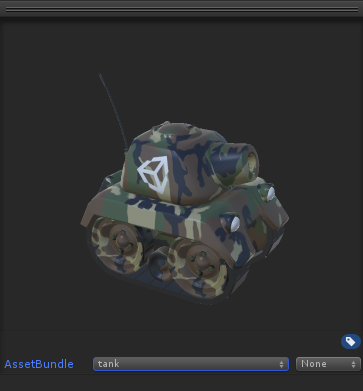


使用AssetBundle名称下拉菜单进行分配资源，在这里要不选择一个现有的要不创建一个新的。 

创建一个新的AssetBundle，选择New并且文本输入框就会有效让你输入名字 如果要移除一个资源，只要选择None就可以 如果想从列表中删除一个AssestBundle名字，那必须要把所有的分配给这个AssestBundle的资源的名字都从AssestBundle中删除掉。然后可以选择Remove Unused Names，将删除所有没被使用的AssestBundle的名字。



AssetBundles名字必须小写，如果不遵循的话，unity会自动处理成小写。

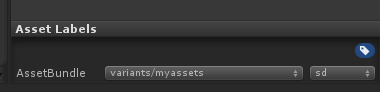
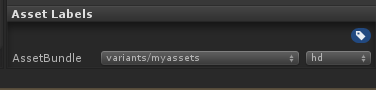
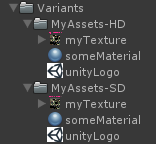


这样AssetBundles名字就被强制改成了小写。

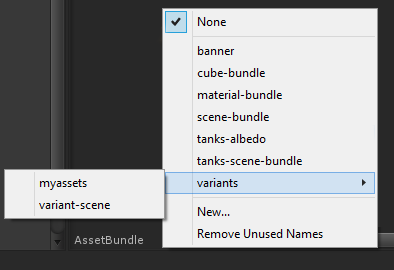
## 使用AssetBundles Variants

允许以各种不同的解决方案解决加载问题，包括下载、存储和更新，一种特定的情况就是AssetBundle 可以依据设备来加载不同的用户偏好等，这是利用AssetBundle Variants。对场景中的一个物体上同一个资源AssetBundle Variants提供不同的Variants，AssetBundle Variants可以实现替换不同的资源到同一个物体上，只有一种Variants可以在任何时间进行加载。

AssetBundle Variants在很多情况上多可以使用，AssetBundle Variants可以为不同分辨率的机器或者高低显卡配置不同的机器或者不同面数的多边形提供相同的资源，AssetBundle Variants可以根据文本、图像、纹理和字体可以为每个受支持的语言不同,地区或主题创建不同的对象。这些资源保存了一系列信息。

以下是AssetBundle的小例子 

在上面例子中，两个文件夹MyAssets-SD和HD同时被分配到myassets的AssetBundle，然后得到一个是别名分别是hd和sd,注意这两个资源具有相同的名称和结构，在创建一个资源的时候父目录被指定到一个AssetBundle上，没有被指到任何一个AssetBundle上。

值得注意的是下面的图片的AssetBundle有一个路径variant/myassets 

这将为名叫myassets的AssestBundle创建一个父菜单，叫variant。一旦一个资源被分配好了之后需要被打包和测试。

一旦资源被指定到 AssetBundle, 这个 AssetBundle 将要被编译和测试。

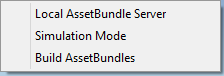
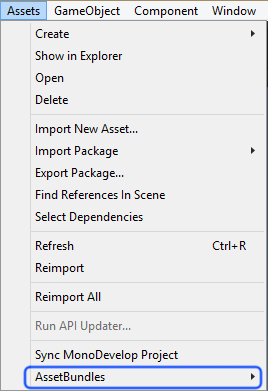
使用 AssetBundle Manager

Unity 提供了直接使用 AssetBundle 的底层 API. 但是这篇教程不会覆盖这些底层 API。关于编译，测试和管理 AssetBundle，这篇教程会专注到 AssetBundle Manager 和它的高层 API 上。

AssetBundle Manager 是一个可下载的，可以安装在当前 Unity 项目中的包，它提供高层 API 和改进了 AssetBundle 的流程。AssetBundle Manager 可以再 这里 下载。要在项目中使用 AssetBundle Manager，简单的将它加到当前的项目的 Asset 文件夹中。

编译和测试 AssetBundle 可能是在开发过程中的一个痛点。资源会时常的改变。使用底层 AssetBunle API 时，测试需要规律的编译和上传 AssetBundle 到一个远程服务器上，然后从当我的项目建立一个网络连接来测试远程服务器上的 AssetBundle。相对于直接操作 AssetBundle 的底层 API, AssetBundle Manager 大幅度的优化了流程。AssetBundle Manager 提供的最核心的功能是一个模拟模式，一个本地的 AssetBundle 服务器和一些快捷的菜单去编译 AssetBundle 和无间隙地和本地 AssetBundle 服务器合作。

把 AssetBundle Manager 加入到项目之后将会在 Asset 菜单中创建一个叫 AssetBundles 的新菜单项。选中 AssetBundles 菜单将会显示一个小菜单选项。



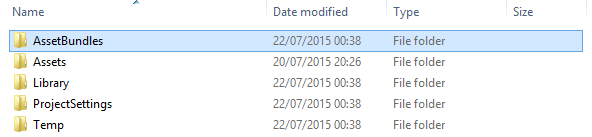
模拟模式开启后，允许编辑器不用实际编译就可以模拟 AssetBundle。要打开模拟模式，选择 Simulation Mode 菜单项。对勾符号表示模拟模式已经开启。要关闭模拟模式就再选择一次菜单项目。然后对勾符号会被移除，模拟模式会被禁用。

当模拟模式开启后，编辑器会查看哪些资源被指定到了 AssetBundle，然后从项目的 hierarchy 中直接使用他们，就像他们在 AssetBundle 中一样。但是这些 AssetBundle 不需要编译。从这点来看，在编辑可以工作到了 AssetBundle 编译后放到远程服务器上一样可以工作。

开启模拟模式最大的好处是，只要资源被正确地指定到 AssetBundle ，在当前运行的项目测试前不需要停下来去编译和重新部署 AssetBundle 就可以修改，操作，导入，删除资源。当开启模拟模式之后，测试是马上生效的。

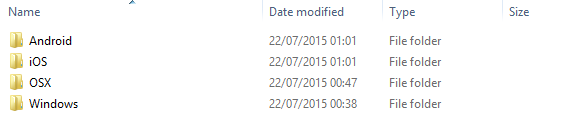
注意 AssetBundle 变体在模拟模式下不支持。测试 AssetBundle 变体，AssetBundle 需要重新编译和部署。但是，本地的资源服务器支持 AssetBundle 变体。

AssetBundle Manager 也可以开启一个本地资源服务器来从编辑器或者本地或移动端的 build 来测试。当本地资源服务器开启后，AssetBundle 必须编译，然后放到项目跟目录中，跟 Assets 文件夹同级的 AssetBundles 文件夹里。



本地资源服务器要求的 AssetBundes 文件夹位置

AssetBundles 本地托管之后, 从当前项目中访问本地资源服务器只需要几行的代码就可以方便的访问。请阅读 AssetBundle 示例项目中的示例，我们会在教程的下面覆盖到。

编译和保存到 AssetBundle 到项目根目录的 AssetBundles 文件夹中可以从 Assets/AssetBundles 菜单中选择 Build AssetBundles 来完成。当 Build AssetBundles 被选择之后， Unity 将会编译所有有资源指定的 AssetBundle, 然后为当前的平台编译和优化他们，最后保存他们和一个主清单到项目根目录下的 AssetBundles 文件夹下。如果没有 AssetBundles文件夹，Unity 会创建一个。在 AssetBundles 文件夹里，AssetBudle 按照编译目标平台来组织。 

“AssetBundles” 文件夹，按照编译目标平台来分组

AssetBundle 被编译后部署到远程服务器或者开始本地资源服务器，这些 AssetBundles 可以在运行期下载和插入到项目中。

## AssetBundle 练习

练习 AssetBundle, 这教程将会使用 AssetBundle Manager。AssetBundle Manager 会应付 AssetBundle 的加载和他们相关的资源依赖。利用 AssetBundle Manager 来从 AssetBundle 中加载资源，脚本需要使用 AssetBundle Manager 提供的 API。

AssetBundle Manager 的 API 包括：

* Initialize() 初始化 AssetBundle 清单对象
* LoadAssetAsync() 从指定的一个 AssetBundle 中加载资源并处理所有的依赖
* LoadLevelAsync() 从指定的一个 AssetBundle 中加载场景并处理所有的依赖
* LoadDependencies() 加载指定的 AssetBundle 的所有独立的 AssetBundle
* BaseDownloadingURL 设置用来自动下载依赖的基本地址
* SimulateAssetBundleInEditor 在编辑器中设置模拟模式
* Vraiants 设置当前的变体
* RemapVariantName() 根据当前的变体决定正确的 AssetBundle

示例文件放置在 AssetBundle Manager 内的 AssetBundle Sample 文件加下。有 3 个基础的示例场景和一个高级的示例场景在 AssetBundleSample/Scenes 文件夹下：

* “AssetLoader” 演示了怎么样从 AssetBundle 加载普通资源
* “SceneLoader” 演示了怎么样从 AssetBundle 加载场景
* “VariantLoader” 演示了怎么样加载 AssetBundle 变体
* “LoadTanks” 更高级，演示了复杂一点的，从同一个场景中加载场景，资源，和 AssetBundle 变体的示例。 每个场景都各个被非常基础的脚本驱动着：LoadAsset.cs，LoadScenes.cs，LoadVariants.cs 和 LoadTanks.cs。

当前重申一下 AssetBundle Manager 提供的流程还是很重要的。

为了能成功的试验 AssetBundle 的使用，这里有三种可能的情景：

第一个情景，没有使用 AssetBundle Manager, AssetBundle 将需要被编译和部署，所有的测试都会在最终完整准备后完成。在这个场景中，每次项目中资源的改变，都需要编译和部署新的 AssetBundle。

AssetBundle Manager 在流程上提供了两个改进。他们是本地资源服务器和模拟模式。

在模拟模式中，编辑器内运行项目时，AssetBundle Manager 会模拟编译后的 AssetBundles。这是使用 AssetBundle 的最快的流程。只需简单的使用 “Assets/AssetBundles/Simulation Mode” 菜单打开 “模拟模式”, 然后测试项目。没有 AssetBundle 会被编译。尽管如此，要注意 AssetBundle 变体在模拟模式下不工作。还有要注意的是，模拟模式开始后，资源可以再项目中操作，并且改变后的效果在 Sence 视图中可以看到，而这使用部署后的 AssetBundle 是不行的。

本地资源服务器提供了部署的 AssetBundle 更精确的演示，但是需要 AssetBundle 被编译和存储到项目中的默认文件夹。当本地资源服务器开启之后，被编译的 AssetBundle 将可以被编辑器和所有运行在本地的，可以通过本地网络连接编辑器的 build 使用。注意这个是能本地测试 AssetBundle 变体的唯一方式。

要运行示例场景，AssetBundle Manager 必须运行在这些模式中的一种。要成功运行 AssetBundle 变体，AssetBundle 必须被编译并且本地资源服务器必须被开启。

### 示例 1：加载资源

使用 “Asset/AssetBundles/Simulation Mode” 菜单打开模拟模式 打开 “AssetBundleSample/Scenes/AssetLoader” 场景 注意场景是个空的只有一个主摄像机，方向光和游戏对象 “Loader” 进入 PlayMode 然后会注意到一个 cube 已经从 AssetBundle 加载到场景里面了 这个场景是被 “LoadAssts.cs” 脚本驱动的。

在脚本编辑器里面打开脚本 “AssetBundleSample/Scripts/LoadAssets.cs”

脚本里有两个公共变量： public string assetBundleName(保存了要被加载的 AssetBundle 的名字) 和 public string assetName(保存了要已加载的 AssetBundle 中加载的资源的名字);这个脚本是由一个 Start() 函数和被 Start() 调用的两个协程组成的。Initialize() 调用了 DontDestoryOnLoad(), 设置了 AssetBundle 的路径和初始化了 AssetBundle 清单。在 InstantiateGameObjectAsync() 中，如果资源不为空，AssetBundleManager.LoadAssetAsync() 调用资源和 AssetBundle 的名字。

重点注意下，在 “AssetBundleSample/Assets” 路径下查看 “MyCube” 资源，会发现 “MyCube” 依赖于 “MyMaterial”，而 “MyMaterial” 依赖于 “UnityLogo”。脚本中只有 “MyCube” 资源被调用，但是所有的依赖资源都被正确的加载了。

AssetBundle 的路径怎么设置也值得注意下。当场景在编辑器中或者从一个开发版 Build 中运行时，这段代码会给本地资源服务器设置 AssetBundle 的位置。（更多关于开发版 Build，请查看发布 Builds 文档。）模拟模式开启后，AssetBundle 会在编辑器中被模拟，这个设置将不会被使用。

对 DontDestoryOnLoad() 作用的理解。虽然在这个非常简单的脚本中并不是绝对需要它，但是他的存在是假设这个脚本会作为一个更复杂的项目的 AssetBundle 加载器基础，它需要在场景变化的以后依然存在。

### 示例 2：加载场景

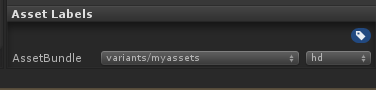
使用 “Asset/AssetBundles/Simulation Mode” 菜单打开模拟模式 开始 “AssetBundleSample/Scenes/SceneLoader” 场景 注意场景是个空的只有一个主摄像机，方向光和游戏对象 “Loader” 开打 PlayMode 然后会注意到一个 cube 和 plane 已经从 AssetBundle 加载到场景里面了 这个场景被 “LoadScene.cs” 脚本驱动着。

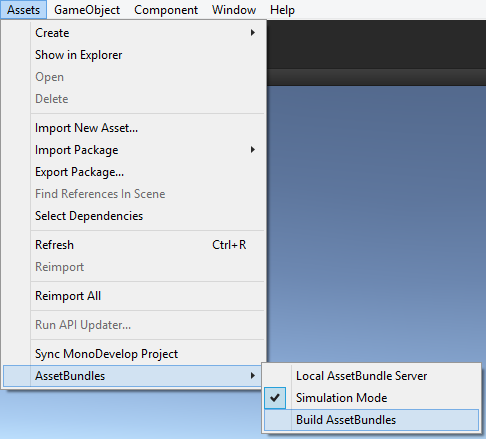
在脚本编辑其中打开 “AssetBundleSample/Scripts/LoadScenes.cs” 脚本。

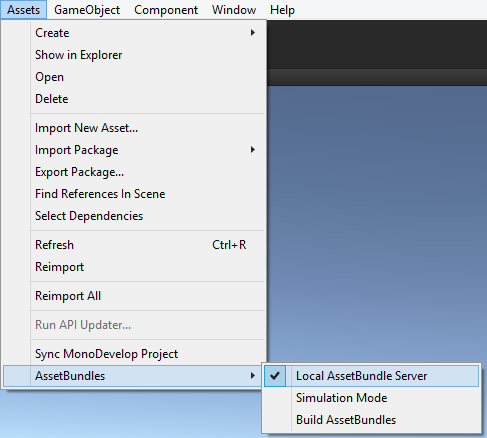
脚本里有个两个公共变量：public string sceneAssetBundle(保持了要加载的 AssetBundle 的名字)和public string sceneName(保持了要从已加载的 AssetBundle 里加载的场景的名字);这个脚本是有一个 Start() 函数和被 Start() 调用的两个协程组成的。Initialize() 调用了 DontDestoryOnLoad(), 设置了 AssetBundle 的路径和初始化了 AssetBundle 清单。在 InitializeLevelAsync() 里使用 AssetBundleManager.LoadLevelAsync() 调用场景名字和 isAdditive 来请求一个场景。如果场景为空，AssetManager 会在控制台显示出错误，然后协程结束。

### 示例 3：变体

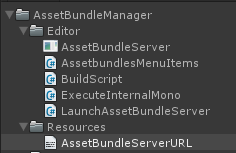
要使用 AssetBundle 变体，需要编译 AssetBundle, 因为模拟模式下不支持它。在编译 AssetBundle 和它的变体钱，确保所有的资源以及被正确地指定 AssetBundle 名字和如果要被 AssetBundle 变体利用到的话，AssetBundle 变体的名字也要指定。



当所有的资源都指定到 AssetBundle 或者 AssetBundle 变体后，选择 “Assets/AssetBundles/Build AssetBundles” 菜单来编译它们。 

默认情况下，AssetBundle 会根据当前的平台优化，并编译进项目跟目录下的 “AssetBundles” 文件夹内，并按平台分组。为了简化流程，不部署新编译出来的 AssetBundle 远程，需要开启本地资源服务器。通过 “Assets/AssetBundles/Local AssetBundle Server” 可以开启本地资源服务器。 

本地资源服务器应该想其他任何网络连接一样受限制，可能是权限需求，防火墙和其他限制。本地资源服务器启动时会被设置到默认 IP 地址和端口的本地资源访问的服务器，通常是 <http://192.168.1.115:7888/>. 这个只是暂时的，它被存储在 Resources/AssetBundleServerURL 文件里面。这些信息会被 AssetBundleManager 设置或自动改变，用户不需要关注它们。



当本地资源服务器运行的时候，编译后 AssetBundle 可以被本地测试。

选择 “Assets/AssetBundles/Simulation Mode” 确保模拟模式被禁用了 选择 “Assets/AssetBundles/Local AssetBundle Server” 确保本地资源服务器开启 打开 “AssetBundleSample/Scenes/VariantLoader” 注意场景是个空的只有一个主摄像机，方向光和游戏对象 “Loader” 退出 PlayMode (如果在 PlayMode 下) 打开 PlayMode 选择 “Load HD” 注意同一个 Cube 和 Sprite 加载进场景了，但是材质和他依赖的独立纹理和 Sprite 纹理却从不同的 AssetBundle 加载。这些材质有不同的颜色，图片有更高的分辨率。 这个场景是被 “LoadVariant.cs” 脚本驱动的。

从编辑器中打开 “AssetBundleSample/Scripts/LoadVariants”。

这个脚本跟 “LoadScenes.cs” 几乎差不多。主要的区别就是用来区别需要加载的 AssetBundle 变体的变量和设置当前变体的代码。还有用来创建 UI 按钮的额外的代码。

public string variantSceneAssetBundle;==>保存要加载的 AssetBundle 的名字

public string variantSceneName;==>保存要从已加载的 AssetBundle 中加载的场景名字

private string[] activeVariants;==>保存用来区分需要加载的 AssetBundle 变体的 AsssetBundleVariantNames

private bool bundlesLoaded;===>用来在加载完资源之后隐藏 UI

脚本由一个 BeginExample() 函数和被 Start() 调用的两个协程组成。 BeginExample() 在 OnGUI()函数中 被 Load HD 或者 Load SD 按钮调用。Initialize() 调用了 DontDestoryOnLoad(), 设置了 AssetBundle 的路径和初始化了 AssetBundle 清单。在 BeginExample() 方法里，在调用 Initialize() 和 InitializeLevelAsync() 之间，当前的变体被设置了。这里被设置的值从靠 OnGUI 里的 “Load HD” 或者 “Load SD” 按钮创建的。在 InitializeLevelAsync() 里使用 AssetBundleManager.LoadLevelAsync() 调用场景名字和 isAdditive 来请求一个场景。如果场景为空，AssetManager 会在控制台显式出错误，然后协程结束。

这里需要重视的是 AssetBundle 变体是怎么样加载的。activeVariants 数组包含了所有可能的 “激活的” 变量名列表。这个数组用来设置 AssetBundleManager.ActiveVariants 属性。当加载一个含有变体的 AssetBundle 时，AssetBundle Manager 将会选择在 ActiveVariants 属性里含有 “激活” 的变体名字的 AssetBundle。当前的示例中，ActiveVariants 属性只包含一个元素。当前的变体要么是 “sd”，要么是 “hd”。在 ActiveVariants 属性中有多个实体是有可能的。比如，可能有下面一些 AssetBundle: my-material.sd，my-material.hd， my-text.english，my-text.catalan，my-text.welsh。ActiveVariants 属性可以包含 “hd” 和 “danish” 两者或者 “sd” 和 “english” 等等任何可以有其他可能组合的变体名。这种方式下，AssetBundle Manager 分开可以加载 hd/sd 图片和语言选择。

有些规则值得注意下。如果，因为一些因素，有一些指定了变体的 AssetBundle，但是在 ActiveVariants 属性里没有 “激活” 的变体名 - 比如当前例子中的 “sd” 或者 “hd” 不在 ActiveVraiants 属性中 - AssetBundle Manager 将会简单的选择第一个它发现的正确名字的 AssetBundle 而忽略变体名。再如果，又因为一些因素，在 ActiveVariants 属性里对同一个 AssetBundle 集合有多个 “激活” 的变体名 - 比如， 在当前的例子里，“sd” 和 “hd” 都在 ActiveVariants 属性里 - AssetBundle Manager 将选择在 ActiveVariants 属性中的第一个变体名。

### 示例 4：坦克示例

这个更复杂的示例将包括这篇文章中的所有内容，包括从 AssetBundle 中加载场景和为分辨率，内容和位置加载 AssetBundle 变体。

选择 “Assets/AssetBundles/Simulation Mode” 确保模拟模式被禁用了 选择 “Assets/AssetBundles/Local AssetBundle Server” 确保本地资源服务器开启 打开 “AssetBundleSample/Scenes/TanksLoader” 注意场景是个空的只有一个主摄像机，方向光和游戏对象 “Loader” 进入 PlayMode 选择一个分辨率，风格和语言 注意在 UI 里的选择项就是加载的资源 如果没有显式的选择一个，AssetBundleManager 将自动选择（基于上面的原则）一个并且在命令行输出一个警告。 场景靠 “LoadTanks.cs” 脚本驱动。

在编辑器里面打开 “AssetBundleSample/Scripts/LoadTanks.cs” 脚本。

这个脚本与 “LoadScenes.cs” 和 “LoadAssets.cs” 非常像。脚本使用代码去加载依赖变体的场景和一样依赖变体的额外的游戏对象。也有一些额外的代码创建 UI 按钮。

* public string sceneAssetBundle; 保存携带场景的 AssetBundle 的名字
* public string sceneName; 保存要从已加载的 AssetBundle 中加载的场景名字。
* public string textAssetBundle; 保存携带文字资源的 AssetBundle 的名字
* public string textAssetName; 保存要从已加载的 AssetBundle 中加载的文字资源的名字
* private string activeVariants; 保存要传给 AssetBundleManager 的 ActiveVariants
* private bool bundlesLoaded; 用来资源加载之后隐藏 UI
* private bool sd, hd, normal, desert, englisth, danish; 保存用来设置 ActiveVariants 的值
* private string tankAlbedoStyle, tankAlbedoResolution, languge; 保存用来设置 ActiveVariants 的值

脚本由一个 BeginExample() 函数和被 Start() 调用的两个协程组成。 BeginExample() 在 OnGUI()函数中 被 “Load Scene” 按钮调用。Initialize() 调用了 DontDestoryOnLoad(), 设置了 AssetBundle 的路径和初始化了 AssetBundle 清单。在 BeginExample() 方法里，在调用 Initialize() 和 InitializeLevelAsync() 之间，当前的变体被设置了。这里被设置的值从靠 OnGUI 里的 “Load Scene” 按钮创建的。在 InitializeLevelAsync() 里使用 AssetBundleManager.LoadLevelAsync() 调用场景名字和 isAdditive 来请求一个场景。如果场景为空，AssetManager 会在控制台显式出错误，然后协程结束。在 InstantiateGameObjectAsync() 中资源和 AssetBundle 名字被 AssetBundleManager.LoadAssetAsync() 调用。如果调用的资源不为空，它会被实例化。如果 AssetBundle 不能被加载或者资源不能被请求，控制台会打印出错误来。

这小结要注意的内容是，多个 资源，AssetBunle 和 AssetBunle 变体怎么被访问和加载进场景里，和怎么样在运行期设置这些值。

## 参考链接

* [原文](https://unity3d.com/cn/node/17559)

## 工程下载

<https://git.oschina.net/dingxiaowei/AssetBundleManager.git>