# 场景管理

## 多场景编辑

多场景编辑允许我们在同一个场景窗口中打开多个场景，而且可以更容易实现运行时的场景管理。

同时编辑多个场景是新一代引擎具有的功能，它可以方便开发者创建巨大的无缝衔接地图，还可以让多人同时编辑同一个场景（比如技术人员、设计师、美术人员分别修改场景中不同的部分），通过拆分场景、合并场景的方法让合作变得非常容易。

本章将介绍和多场景管理有关的几个问题：

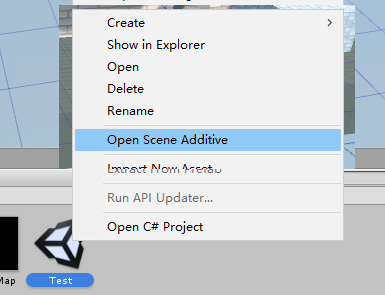
1、多场景编辑。

2、运行时场景管理，SceneManager类。

3、其它注意事项。

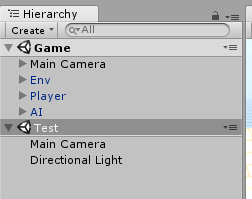
### 在编辑器中打开多个场景

要打开一个新的场景并且附加到当前的层级窗口中，有两种方法。一是在工程窗口中的场景文件上点击右键，选择Open Scene Additive（增量打开场景）；二是把工程窗口中的场景拖拽到层级窗口中。



（双击场景会直接切换场景，而选择Open Scene Additive则会将场景附加到层级窗口中，或者把场景文件拖拽到层级窗口也有同样的效果）

在场景中打开了多个场景后，每个场景在层级窗口中都是一个独立的父节点。且每个场景会有一个单独的“分隔栏”，在这条分隔栏里可以看到每个场景的标题、是否保存过（\*号代表有未保存的改动）。

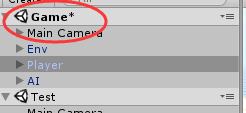


（同时打开多个场景的示例，Unity图标的那一行就是该场景的栏位）

当场景在层级窗口中打开以后，还可以将场景卸载（Unload）或加载（Load），甚至还可以隐藏/显示整个场景中所有物体。这些操作与直接关闭、打开场景是不同的。

当打开的场景较多时，可以使用折叠的方式方便查看所有场景，这个操作类似父物体折叠。

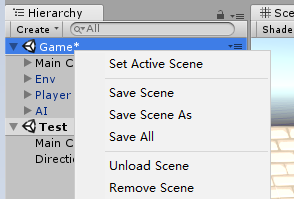
当编辑多个场景时，每个场景都可以单独编辑、单独保存，所以也允许修改多个场景而暂时不保存。这相比每次编辑一个场景要方便得多。



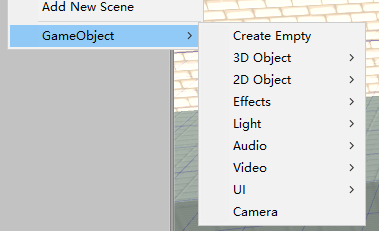
（场景有未保存的修改时，会以\*号标记）

每个场景都可以单独保存，在每个场景名称栏的右键菜单中即可选择。另外，选择主菜单中的Save Scene或者按下Ctrl/Cmd + S键可以同时保存所有场景。

### 场景分隔栏菜单



|  |  |
| --- | --- |
| Set Active Scene | 设置此场景为激活状态。激活后，新建的物体会进入这个场景。同一时间有且只有一个场景处于激活状态。 |
| Save Scene | 单独保存这个场景。 |
| Save Scene As | 将这个场景另存为一个场景文件。 |
| Save All | 保存所有场景。 |
| Unload Scene | 卸载场景，但是它还会保留在层级窗口中。 |
| Remove Scene | 卸载场景，并从层级窗口中删除。 |
| Select Scene Asset | 在工程窗口中选中这个场景对应的文件。 |
| GameObject | 新建游戏物体的快捷入口。 |



（在场景分隔栏的菜单中也可以创建游戏物体）

### 多场景烘培光照贴图

要为多个场景同时烘培光照贴图（Lightmap），只需要打开所有需要操作的场景，在灯光窗口（Lighting）中关闭自动（Auto）选项，点击“Generate Lighting”按钮即可。

照明计算的输入数据是来自所有场景的静态物体和灯光。 而阴影和GI（全局光照）光线反弹将适用于所有场景。 但是，光照贴图和实时GI数据会为每个独立场景分离。这意味着场景之间的光照贴图永远不会共享，卸载场景时可以安全卸载它们。光照探针（Lightprobe）数据总是共享的，并且所有场景烘培在一起的所有探测器都在同一时间加载。

另外，在编辑器专用脚本中，可以使用Lightmapping.BakeMultipleScenes方法来自动为多个场景烘培光照信息。

### 多场景烘培寻路网格

要为多个场景同时烘培寻路网格（Navmesh），可以打开所有要处理的场景，然后在寻路窗口中点击烘培（Bake）按钮即可。所有的寻路数据将会被烘培到一个文件里，所有相关场景都共享这一份寻路信息。但是注意，这个寻路信息文件的位置会和当前激活的场景放在一起（例如当前激活的是Main场景，则寻路信息文件位于Main/NavMesh.asset），所有已加载的场景会共享这一份寻路信息。在烘培寻路信息之后，应当确保这些相关的场景都被保存，且要保证之后这些场景能找到寻路信息文件。

另外，在编辑器专用脚本中，可以使用NavMeshBuilder.BuildNavMeshForMultipleScenes函数来自动进行上述操作。

### 多场景烘培遮挡剔除信息

要为多个场景同时烘培遮挡剔除信息，同样只要打开所有需要操作的场景，然后再遮挡剔除窗口（Occlusion Culling Window）中点击烘培按钮即可。遮挡剔除信息会保存在Unity工程的Library/Occlusion目录中，且每个场景都会记录到剔除信息文件的引用。所以在烘培之后应当保存一次所有场景。

在以附加到当前场景的方式打开一个场景时，如果新的场景和原有场景引用同一份遮挡剔除数据，那么遮挡剔除系统的行为就像是对整个合并后的场景应用了遮挡剔除。

### 多场景运行游戏

如果在运行游戏时有多个场景存在于层级窗口中，就会自动添加一个新的场景叫做“DontDestroyOnLoad”。

Unity5.3之后的版本提供了这个新的特性，所有被标记为“DontDestroyOnLoad”的物体，可以认为并不是属于某个单独场景的，它们会在场景切换后依然存在。这些被标记的物体会出现在那个特殊的DontDestroyOnLoad场景中，可以很方便地选择和操作。

### 场景相关设置

有一些设置是针对单独的场景的，包括：

1. 渲染设置（Render Settings）和光照贴图设置（Lightmap Settings），二者都在光照窗口（Lighting Window）中。
2. 导航网格设置。
3. 遮挡剔除窗口中的场景设置。

这些设置的工作方式是，每个场景都可以单独设置和保存各自的设置信息，且保存在场景文件中。场景之间是独立的。

如果打开了多个场景，那么这时设置的渲染、寻路信息实际上是针对激活的场景的。也就是说，如果要改变一个场景的相关设置，要么单独打开这个场景，要么在多场景编辑中将它激活，才可以对那个场景进行设置。

当在编辑器中激活其它场景，或者在游戏运行时切换场景时，原有场景的设置信息也同样会被替代，换成新场景的设置

### 注意事项

在文件菜单中，“Save Scene”选项会保存所有场景，而“Save Scene As”选项则只保存当前激活的场景。

当拖拽一个场景文件到层级窗口时，默认会加载那个场景。而如果按住Alt键进行拖拽，就不会立即加载它，而是保留未加载状态。

在工程窗口中新建场景的方法类似于创建脚本文件，都是在菜单中进行操作。

如果经常要进行多场景编辑，有时我们想保存多个场景打开的状态，否则每次关闭Unity都必须重新打开多个场景。针对这个问题，Unity提供了编辑器脚本方法EditorSceneManager.GetSceneManagerSetup用于获取当前编辑器中的场景列表。可以序列化这个列表并保存到文件中，下次想要恢复的时候，就可以使用EditorSceneManager.RestoreSceneManagerSetup方法还原场景列表。

获得场景列表后，可以通过它的sceneCount属性获得列表长度，并用GetSceneAt方法获得某一个序号的场景。

GameObject.scene方法，可以获得一个游戏物体所在的场景。

SceneManager.MoveGameObjectToScene方法可以移动一个物体到某个场景中。

使用SceneManager.LoadScene(“场景名称”, LoadSceneMode.Additive)方法来加载场景，并使用SceneManager.UnloadScene方法来卸载场景，这样就可以实现基本的场景管理操作。本章还会继续介绍有关运行时场景管理的方法。

最后还有一个问题：跨场景的物体引用关系。在制作游戏的过程中，不可以跨场景引用物体，比如场景A的某个物体的脚本中，使用变量引用了场景B的某个物体。而在游戏运行时这种引用是可能的，因为加载了多个场景时，这些场景是在同一个游戏环境中。

## 运行时场景管理

### 场景管理类SceneManager

SceneManager类是脚本中，在运行时管理场景的类。先大致介绍一下它的所有属性、方法和事件。

1. **属性：**

|  |  |
| --- | --- |
| sceneCount | 已加载（loaded）的场景总数 |
| sceneCountInBuildSettings | 工程设置中的场景总数 |

1. **方法**

|  |  |
| --- | --- |
| CreateScene | 用指定的名称创建一个新的空场景 |
| GetActiveScene | 获得当前激活的场景 |
| GetSceneAt | 用下标获得某个已加载的场景，下标是工程设置中场景列表的下标 |
| GetSceneByBuildIndex | 用下标获得场景结构体 |
| GetSceneByName | 以场景名称获取某个已加载的场景 |
| GetSceneByPath | 以场景路径获取某个已加载的场景 |
| LoadScene | 以下标或场景名称加载场景 |
| LoadSceneAsync | 加载场景的异步方法。在后台异步加载场景，通常和协程配合使用 |
| MergeScenes | 融合一个场景到另一个场景中 |
| MoveGameObjectToScene | 将一个游戏物体从一个场景移动到另一个场景 |
| SetActiveScene | 激活一个场景 |
| UnloadSceneAsync | 异步卸载场景。会销毁场景中的所有物体并移除此场景 |

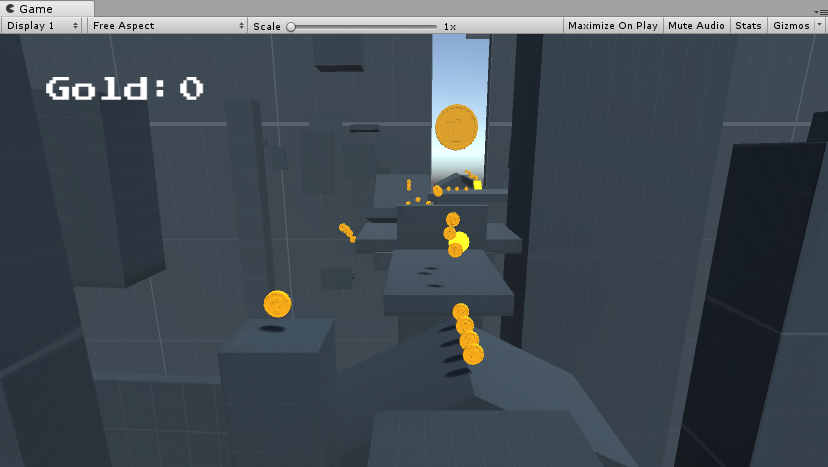
1. **事件**

|  |  |
| --- | --- |
| activeSceneChanged | 订阅这个事件，当激活场景时得到通知 |
| sceneLoaded | 订阅这个事件，当场景加载完成时得到通知 |
| sceneUnloaded | 订阅这个事件，当场景卸载完成时得到通知 |

虽然SceneManager类的方法看似很多，实际上最常用的方法就是加载场景LoadScene以及异步版本的加载场景LoadSceneAsync。

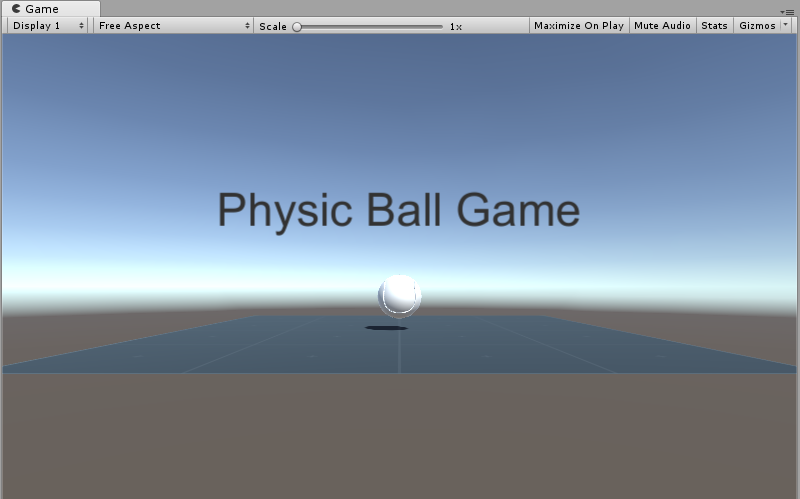
### 运行时切换场景

下面我们用一个例子来演示游戏中基本的场景切换。选任意一个能够独立运行的游戏场景做为例子，可以是你自己制作的简单游戏场景：



（本文以一个简单的3D物理滚球游戏场景为例）

游戏场景名称为Game，直接运行就可以开始游戏了。我们现在考虑为它制作一个初始画面：



象征性的用游戏素材制作了开始场景，命名为Start。这样就和很多游戏一样，带有一个开始画面。

我们的目标是实现简单的场景切换，现在添加一个用于切换场景的脚本ChangeScene.cs：

|  |
| --- |
| using UnityEngine;  using UnityEngine.SceneManagement;  public class ChangeScene : MonoBehaviour {  // 在编辑器中指定场景名称或者场景序号，使用其中一种即可  public string changeSceneName;  //public int changeSceneIndex;  void Update () {  // 如果按下键盘T键  if (Input.GetKeyDown(KeyCode.T))  {  使用LoadScene方法，通过名称或者序号加载场景  SceneManager.LoadScene(changeSceneName);  //SceneManager.LoadScene(changeSceneIndex);  }  }  } |

在Start场景中创建一个空游戏物体，命名为“ChangeScene”，并挂载上面的脚本。挂载成功后在检视窗口中，输入要切换的场景的名称，比如这个例子里要切换到“Game”场景：

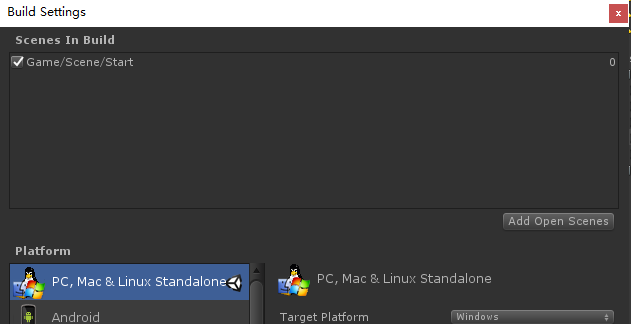


之后在Start场景中运行游戏，按T键，会报告以下错误：

|  |
| --- |
| Scene 'Game' couldn't be loaded because it has not been added to the build settings or the AssetBundle has not been loaded.  To add a scene to the build settings use the menu File->Build Settings...  UnityEngine.SceneManagement.SceneManager:LoadScene(String)  ChangeScene:Update() (at Assets/Game/Scripts/ChangeScene.cs:14) |

意思是说Game场景不存在，应当在Build Settings（打包设置）窗口中添加相应的场景。这个提示非常详细了，按照它的提示操作。

在主菜单中选择File > Build Settings，打开设置窗口：



之后可以用拖拽的方式，将场景文件从工程窗口中拖动到场景列表中即可。其中，场景的序号就是场景在这个列表中的序号，场景以文件路径的方式显示出来，所以我们可以用序号、名字或者路径来代表一个场景。

将两个场景都添加好以后，再测试，会发现能够正常切换场景了。

切换场景的关键有两点：

1. 场景管理器SceneManager的使用。
2. 所有游戏中用到的场景，一定要在Build Setting窗口中进行设置。

### 场景切换时不销毁游戏物体

在8.1节中，提到过“Don’t destroy on load”这个概念。这个功能意思是在切换场景时，不销毁某些物体。它是为了解决什么问题而设计的呢？

在某些游戏中，游戏世界是由许多小场景组成的，比如传统的RPG游戏，主角的家是一个二层小楼，包含主角卧室、父母卧室、客厅、地下室、厨房、阳台这6个房间。这些房间是通过楼梯、门连接起来的。而在游戏制作时，就可以将这些房间全部看成是独立的场景，玩家走上楼梯、进门都可以通过切换场景来做到。

在这个设计中，有一点非常重要——在切换场景时，玩家角色本身是不需要被销毁的，相对的，每个场景在制作时中也不能包含玩家角色。玩家独立于场景之外，在场景之间游走。

要做到这一点，只需要在玩家身上的脚本中写上：

|  |
| --- |
| DontDestroyOnLoad(gameObject); |

DontDestroyOnLoad是一个静态方法，只要参数是主角的游戏物体，即可让该主角在场景切换时不销毁。而默认主角也是场景中一个普通的物体，会在切换场景时消失。

如果使用这个方法，有几个问题需要注意：

1. 如果某个角色不被销毁，那么他就应当只被创建一次。
2. 物体不销毁，但是在不同场景中的位置不同，要在场景初始化时重新摆放物体的位置。
3. 初学者常见问题：每个场景中都摆放了主角，但是主角被标记为DontDestroyOnLoad。这样的结果就是会在场景切换时出现多个主角。

在设计多场景游戏时，只要考虑清楚哪些物体不该被销毁，之后总是采用相同的策略，就能避免很多问题。

### 异步加载场景

前面使用的SceneManager.LoadScene方法有一个比较严重的缺陷。首先，加载场景的函数调用会立即执行加载工作，导致游戏会一直等待LoadScene函数执行完毕；其次，场景的大小是不一定的，如果场景非常大，那么加载时间就会比较长。

所以问题就是，在切换场景时，玩家会感受到明显的等待或卡顿，甚至在某些设备上出现“停止响应”的问题。

因此实际游戏开发中，更多时候我们会专门设计一个读取界面（Loading界面）。然后在后台加载场景时，Loading界面依然可以播放动画，整个游戏也不会完全停滞。这里就要用到Unity提供的LoadSceneAsync方法来实现这个功能了。

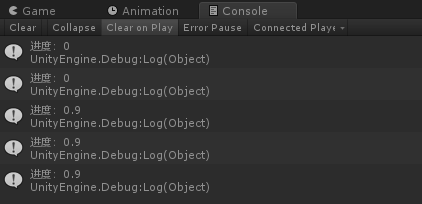
类似LoadSceneAsync这样的“异步”方法还有很多，比如发送/接收网络消息、加载资源都可以是异步的，可以改善用户体验。这种异步方式的实现原理离不开Unity的协程（Coroutine）支持。下面将之前的例子改为异步调用：

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.SceneManagement;  public class ChangeScene : MonoBehaviour {  // 正在加载场景的标记，防止重复加载  bool isLoadingScene = false;  public string changeSceneName;  //public int changeSceneIndex;  void Update () {  if (Input.GetKeyDown(KeyCode.T))  {  if (!isLoadingScene)  {  // 开启协程  StartCoroutine(ChangeSceneAsync());  }  }  }  IEnumerator ChangeSceneAsync()  {  isLoadingScene = true;  AsyncOperation async = SceneManager.LoadSceneAsync(changeSceneName);  while (!async.isDone)  {  Debug.Log("进度：" + async.progress);  yield return null;  }  }  } |

简单解释一下上面的代码。当按下T键以后，开启了一个协程，协程在第一次调用就执行了LoadSceneAsync方法，但是这个方法只是开始了Unity引擎内部加载场景的任务，并不等待加载完成，所以代码继续执行，直到第一个yield return的位置，本函数暂时中止，让出执行时间给其他函数。下一帧会再回到yield return的位置继续执行。

这样就达到了加载场景时，游戏不会在函数内暂停的目的。

另外，使用AsyncOperation的isDone属性来判断加载工作是否完成，使用async.progress属性可以获得大致的加载进度，取值范围0~1.0。上述代码在加载场景时打印了加载进度到控制台：



场景比较简单的话加载进度不够均匀，可以自己设计一种方法来获得更准确的进度百分比。