第 5 章

CHAPTER 5

Unity3D 插件扩展

5.1 插件扩展的价值

当用户需要实现某个功能，但是市面上没有完全支持的功能，却恰好有相似的插件能满足这个功能的部分需求时，是否会选择以此插件为基础扩展后使用呢？当某项目已经使用一个插件迭代了很多版本，但最近新需求的实现需要要么换成新插件，要么对现有插件进行扩展会如何选择呢？再例如，当某项目使用的开源插件由于 Unity 引擎版本的升级导致一些 API 无法使用，这时是选择换掉这个插件还是修改这些 API 呢？如果扩展插件带来的效益更高，则这一定会成为不二的选择。

扩展 Unity 插件的价值不仅体现在提高开发效率和游戏性能上，还在于能促进个性化开发、实现特定功能及推动技术创新。

首先，扩展 Unity 插件可以极大地提高开发效率。通过对现有插件进行扩展，开发者能够在避免重复造轮子的情况下，快速满足项目的特定需求。例如，一个基于 Unity 的 XR 开发插件可能已经提供了基础的 XR 功能，但通过扩展，开发者可以加入特定的图像识别和交互功能，以满足特定场景的需求。这种扩展不仅节省了开发时间，也提高了项目的完成质量。

其次，Unity 插件的扩展对于满足特定项目需求也是不错的解决办法。标准的插件往往是无法完全满足这些特定需求的，但通过自定义和扩展插件，开发者可以为插件添加独特的功能，这不仅能增加游戏的个性化，还能提高用户体验。例如 UGUI 的 Button 组件本身没有单击后保持按下状态的功能，但通过扩展可以实现基于 Button 扩展一个 KeepButton 的按钮。

最后，开发者在扩展插件的过程中，可能也会探索和实验新的技术或方法。这些创新不仅可以解决特定的开发挑战问题，也可能引领新的技术趋势，为行业带来新的视角和思路，推动技术创新。

总而言之，不论是插件的开发团队，还是插件的用户，如果插件扩展能提高开发效率并且满足特定需求，则这种扩展就具有较高的价值。

5.2　如何扩展现有插件

扩展现有插件分为两种情况，一种是拥有插件源码（插件开发团队成员或开源插件），另一种是没有插件源码。在有源码的情况下，但凡涉及修改源码时都要考虑是否会破坏原有功能。除此之外，其他的步骤是一样的，流程如图 5-1 所示。

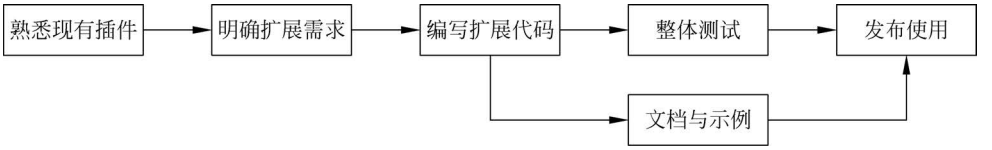

图 5-1　插件扩展流程

首先，需要了解插件的架构、代码和功能。这可以通过阅读插件文档、研究源码（如果可以）和 API 来了解。这一步能有效地确保扩展时不会破坏插件原有的功能。

其次，需要明确扩展的内容是什么，这要进一步熟悉与扩展内容相关的功能模块。通常这一步可以通过画图的方式来梳理，这一步完成后就能明确待扩展的内容要通过什么方式接入现有插件。

然后就可以编写扩展代码了，这可能涉及直接修改插件的源代码（如果可以），或者更常见的是，创建新的脚本或模块与原插件进行交互。在这个过程中，保持代码清晰和模块化非常重要，这样不仅有助于维护扩展后的插件，也使代码更容易被理解和使用。

最后，对扩展插件进行测试，以不会干扰原插件功能且能覆盖扩展的需求为目标，通常采用冒烟测试或系统测试等方法覆盖所有功能。测试完成后，需要附加扩展部分的文档说明和使用示例。

这一切准备就绪后，插件便可以发布给开发者使用，但需要注意的是，如果扩展的是具备他人版权的插件，则是不能商业化的。

5.3　实例分析：扩展资源管理插件

Addressables 是 Unity 官方推出的用于资源管理的可寻址资源系统。它是基于 Asset Bundle 开发的，提供了异步加载、依赖管理及内存管理等丰富的资源管理功能，也能够让开发者更便捷地实现资源热更新，但 Addressables 无法灵活地按优先级进行下载，本节将基于此需求扩展 Addressables 插件。

首先，Addressables 是一个开源项目，因此是可以修改源码的，但此插件通过 Package Manager 导入后便能保护代码不被轻易改动，本节扩展也以不修改源码为前提，因此可以不用考虑改动会影响原有功能的情况。

其次，明确具体的扩展需求，其实也就是将需求拆分为局部步骤，第 1 步需要定义资源

优先级，在资源加载请求时允许资源根据重要性被标记为不同程序的优先级；第 2 步需要创建一个管理资源加载请求的优先级队列，确保高优先级的资源加载请求能够被优先处理；第 3 步需要根据当前系统负载和资源加载情况，动态调整并行加载任务的数量。

最后，将拆分的小步骤逐一编码实现。

因此，先创建一个枚举，用来表示资源优先级，代码如下：

```
//第 5 章 //AssetPriority.cs

//资源优先级
public enum AssetPriority
{
    High = 0,
    Medium,
    Low
}
```

接下来，创建一个优先级管理器，用于管理和调度资源加载请求。先定义一个下载资源的请求类，包含资源的下载键值、优先级和下载完成的回调。再添加一个公开方法，以便将下载任务加入下载队列，并将队列里的任务按优先级排序后再调用 Addressables.LoadAssetAsync 下载资源，代码如下：

```
//第 5 章 //PriorityLoader.cs

public class PriorityLoader
{
    //< summary >
    //加载请求
    //</summary >
    private class LoadRequest
    {
        public string key;
        public AssetPriority Priority;
        public System.Action < AsyncOperationHandle > OnComplete;
    }

    //加载队列
    private List < LoadRequest > taskQueue = new List < LoadRequest >();

    //< summary >
    //添加资源加载请求
    //</summary >
    //< param name = "key"></param >
    //< param name = "priority"></param >
```

```
    //< param name = "onComplete"></param >
    public void AddLoadRequest < T >(string key, AssetPriority priority, System.Action
< AsyncOperationHandle > onComplete)
    {
         taskQueue.Add(new LoadRequest { key = key, Priority = priority, OnComplete =
onComplete });
        //根据优先级排序
        taskQueue.Sort((x, y) => y.Priority.CompareTo(x.Priority));
        //处理下一个加载请求
        ProcessNext < T >();
    }

    //处理加载请求
    private void ProcessNext < T >()
    {
        if (taskQueue.Count == 0) return;

        var request = taskQueue[0];
        taskQueue.RemoveAt(0);

        AsyncOperationHandle handle = Addressables.LoadAssetAsync < T >(request.key);
        handle.Completed += (op) =>
        {
            request.OnComplete?.Invoke(op);
            //完成后继续处理下一个请求
            ProcessNext < T >();
        };
    }
}
```

如此便完成了对 Addressables 按优先级进行下载的扩展功能。接下来需要测试扩展功能是否生效，需要先将测试资源配置到 Group 中，如图 5-2 所示。

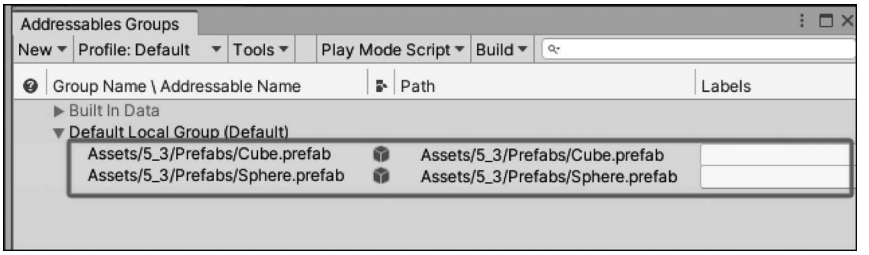

图 5-2　资源分组

然后通过代码加载它们，代码如下：

```
//第 5 章 //Script_5_3.cs

PriorityLoader priorityLoader = new PriorityLoader();
//添加一个低优先级的资源加载请求
priorityLoader.AddLoadRequest<GameObject>("Assets/5_3/Prefabs/Cube.prefab", AssetPriority.Low,
(handle) =>
{
    Debug.Log("Cube 下载完成");
    GameObject cube = GameObject.Instantiate<GameObject>((GameObject)handle.Result);
});

//添加一个高优先级的资源加载请求
priorityLoader.AddLoadRequest<GameObject>("Assets/5_3/Prefabs/Sphere.prefab", AssetPriority.
High, (handle) =>
{
    Debug.Log("Sphere 下载完成");
    GameObject sphere = GameObject.Instantiate<GameObject>((GameObject)handle.Result);
});
```

代码优先加载的是 Cube 预制件，但是优先级被设置为低优先级，而 Sphere 预制件虽然后加载，但是优先级被设置为高优先级。代码执行后可以发现结果是 Sphere 先完成加载，扩展功能生效，如图 5-3 所示。

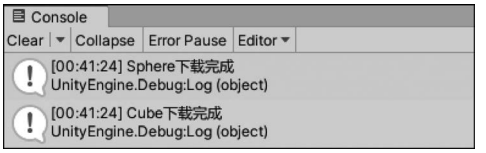

图 5-3　资源根据优先级下载结果