【Unity基础教程】重点知识汇总

（十五）

Unity实现数据持久化的三种常见方法

数据持久化是什么（概念）

概念：数据持久化指的是将应用程序运行时的**数据**存储到**文件、数据库、云端或其他存储介质**中，使得这些数据可以在程序**重启或关闭**后**仍然存在并且可以恢复使用**。这种技术确保了应用程序数据的**持续性**和**稳定性**。简单来说，数据持久化的目的是在程序运行生命周期之外保存数据。

在Unity中，数据持久化通常用于**保存游戏状态、用户设置、进度、玩家分数**等信息。它使得玩家能够在关闭应用后，重新启动游戏时，恢复到上次退出时的状态。

用途及重要性：

- **保存玩家进度：**玩家可以在不同的会话之间保存和恢复进度，**避免**每次启动游戏时都要**从头开始**。
- **配置管理：**用户偏好设置（如音量、画面设置、控制选项等）可以通过数据持久化保存，确保每次启动都能恢复。
- **跨平台同步：**某些游戏可能需要在**不同设备**之间同步进度和设置，持久化的数据可以帮助实现这一点。
- **游戏存档：**游戏的不同存档（如关卡进度、得分、道具等）通常都需要持久化存储。
- **性能提升：**持久化存储**避免**了每次启动时**重新计算**所有数据的过程，可以提升加载和启动速度。

方法一：PlayerPrefs（概念）

概念：PlayerPrefs是**Unity自带**的简单持久化方案，适用于保存**少量**的**键值对**数据，如玩家的设置（音量、分数等）。适合小型数据（如游戏设置、分数、用户首选项等）。

优点：

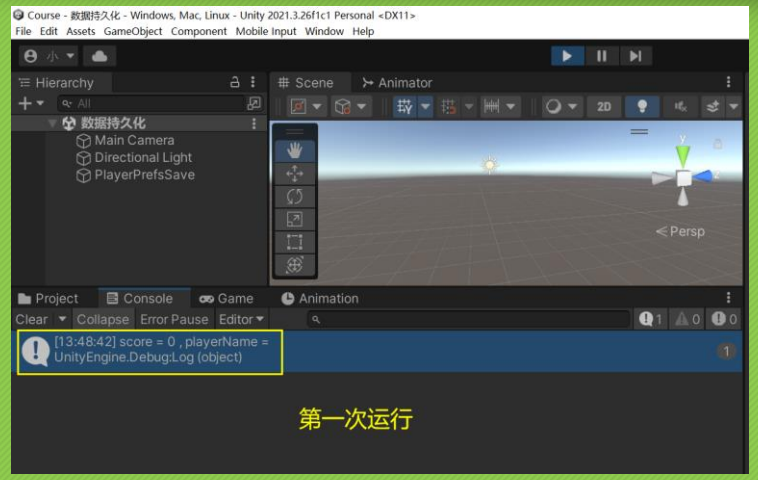
- **使用简单。**内置支持，无需额外库或工具。直接保存数据为键值对。适合保存少量数据。
- 数据存储在**注册表**（Windows）或**偏好设置**（macOS）中，方便读取，跨平台兼容。

缺点：

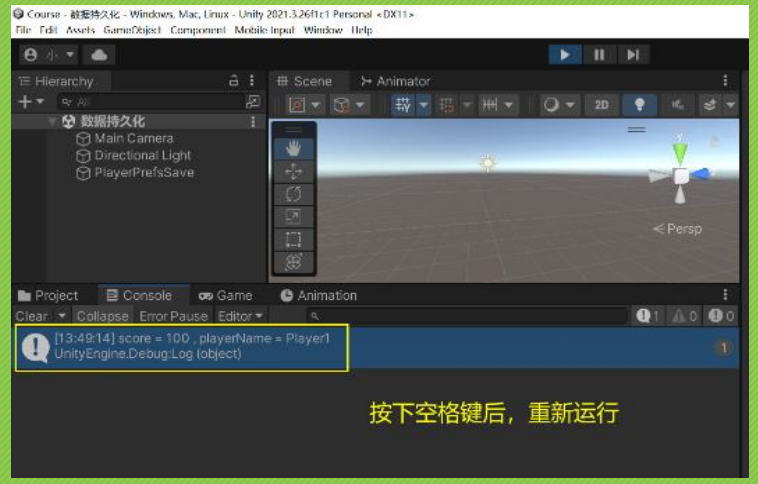
- 数据是以**纯文本**形式存储的，容易被用户**篡改**。
- 存储数据量有限。不适合存储大量复杂数据（如大量的游戏状态）。
- 数据类型有限（**只能存储字符串、整数和浮点数**），无法存储复杂对象。**性能较低**，不适合频繁读写操作。

方法一：PlayerPrefs（具体实现）

```csharp
public class PlayerPrefsSave : MonoBehaviour
{
    void Start()
    {
        // 读取数据
        int score = PlayerPrefs.GetInt("HighScore");
        string playerName = PlayerPrefs.GetString("PlayerName");
        // 在控制台输出
        Debug.Log($"score = {score} , playerName = {playerName}");
    }
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            // 保存数据
            PlayerPrefs.SetInt("HighScore", 100);
            PlayerPrefs.SetString("PlayerName", "Player1");
            PlayerPrefs.Save();
            // 保存提示
            Debug.Log("保存成功!");
        }
    }
}
```

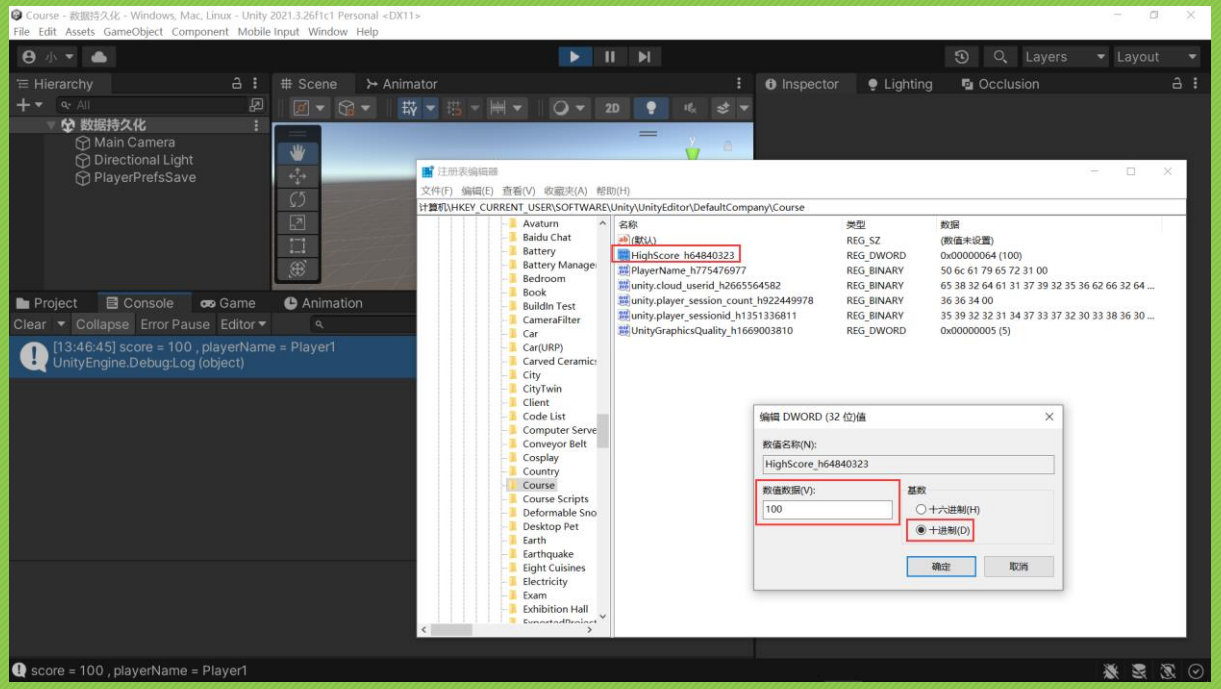


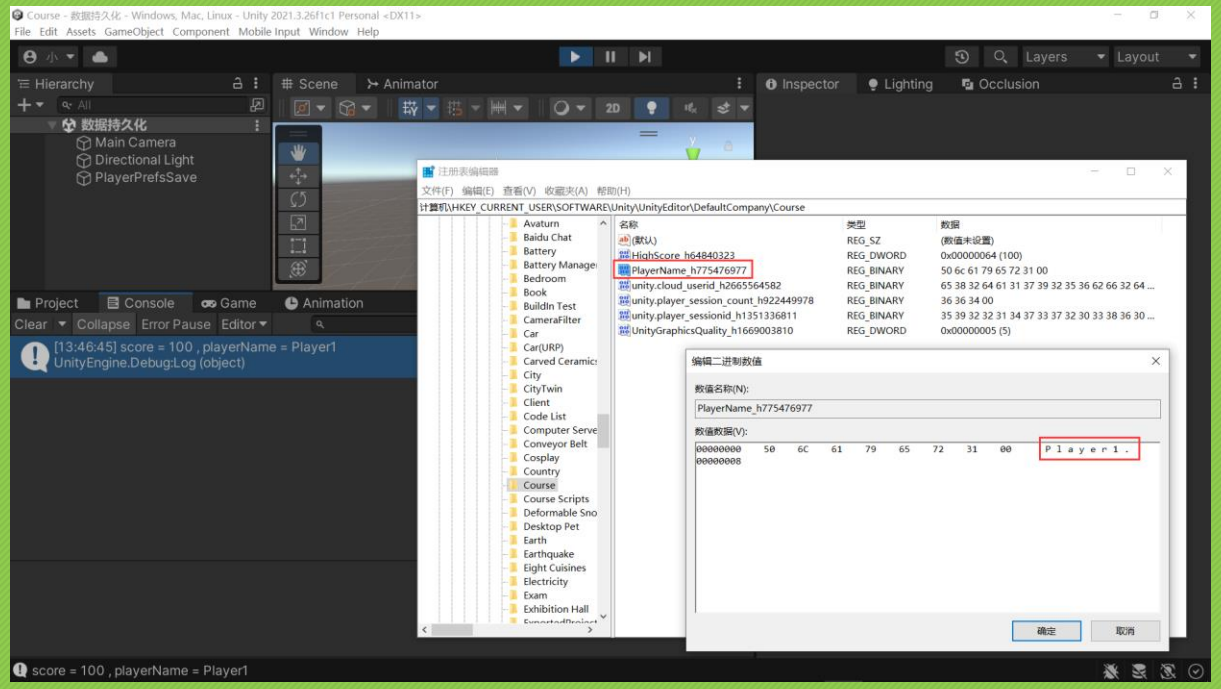
第一次运行



按下空格键后，重新运行

方法一：PlayerPrefs（数据保存位置）





以Windows为例：Win + r 输入 regedit 进入注册表界面。编辑器下保存路径：计算机\HKEY_CURRENT_USER\SOFTWARE\Unity\UnityEditor\CompanyName\ProjectName，打包之后保存路径：计算机\HKEY_CURRENT_USER\SOFTWARE\CompanyName\ProjectName（CompanyName与ProjectName在Unity中Editor-Project Settings-Player界面顶部）

方法二：JSON文件（概念）

概念：将数据以**JSON**的形式保存到文件中。这是比较灵活且常见的持久化方法，非常适合保存复杂的数据结构，如玩家状态、场景信息等。在Unity中使用**JsonUtility**将对象**序列化**为JSON格式，存储在文本文件中。（也有将数据以XML或二进制纯文本的形式保存到文件中）

优点：

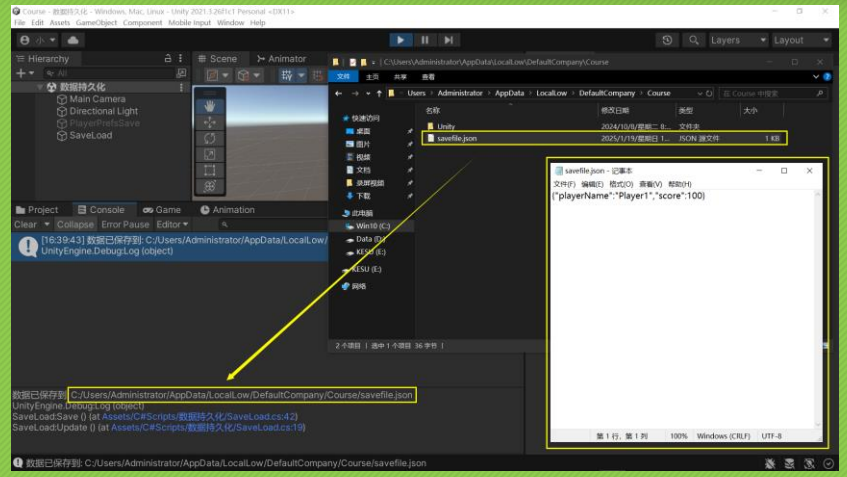
- 更加**灵活**，支持复杂数据结构，数据格式易于理解。
- 数据可被编辑，可实现**跨平台**的数据交换。数据可以**加密**，安全性更好。

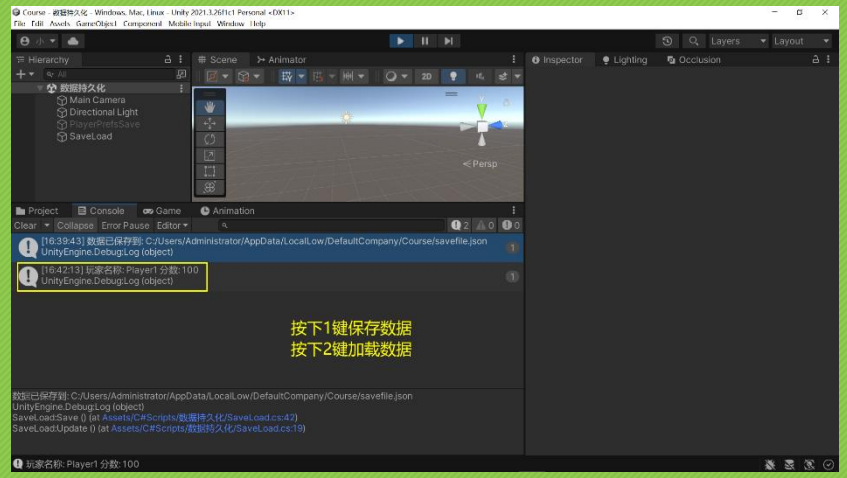
缺点：

- 相比PlayerPrefs，文件存储需要更多的代码来管理文件读写，**复杂度稍高**。
- 对于频繁的读写操作和处理大型数据时，可能**会影响性能**。

方法二：JSON文件（具体实现）

```csharp
public class SaveLoad : MonoBehaviour
{
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            // 保存数据
            Save();
        }
        if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            // 加载数据
            Load();
        }
    }
    /// <summary>
    /// 保存数据到文件
    /// </summary>
    private void Save()
    {
        // 创建一个新的玩家数据对象
        GameData data = new GameData();
        // 设置玩家名称
        data.playerName = "Player1";
        // 设置玩家分数
        data.score = 100;
        // 将数据对象转换为JSON字符串
        string json = JsonUtility.ToJson(data);
        // 将JSON字符串写入文件（文件路径为持久化数据路径/savefile.json）
        File.WriteAllText(Application.persistentDataPath + "/savefile.json", json);
        Debug.Log("数据已保存到: " + Application.persistentDataPath + "/savefile.json");
    }

    /// <summary>
    /// 从文件加载数据
    /// </summary>
    private void Load()
    {
        // 检查保存文件是否存在
        if (File.Exists(Application.persistentDataPath + "/savefile.json"))
        {
            // 从文件中读取JSON字符串
            string json = File.ReadAllText(Application.persistentDataPath + "/savefile.json");
            // 将JSON字符串转换回数据对象
            GameData data = JsonUtility.FromJson<GameData>(json);
            // 输出加载的数据
            Debug.Log("玩家名称: " + data.playerName + " 分数: " + data.score);
        }
        else
        {
            Debug.LogWarning("存档文件不存在！");
        }
    }
}
```






```csharp
public class GameData
{
    // 玩家数据类，用于存储玩家信息
    public string playerName; // 玩家名称
    public int score;         // 玩家分数
}
```

方法三：ScriptableObjects（概念）

概念：ScriptableObject是**Unity特有**的一种数据持久化方式（数据容器，并不直接适用于运行时动态数据的持久化），适合存储在游戏中的**静态数据或配置信息**等**不需要频繁修改**的数据，如角色属性、游戏关卡设置等。

优点：

- Unity内置，适合静态配置数据。
- 易于在编辑器中进行管理和修改，作为**数据模板**。
- **性能高**，**适合频繁读取**的数据。不适合频繁写入的场景（如实时存档）。

缺点：

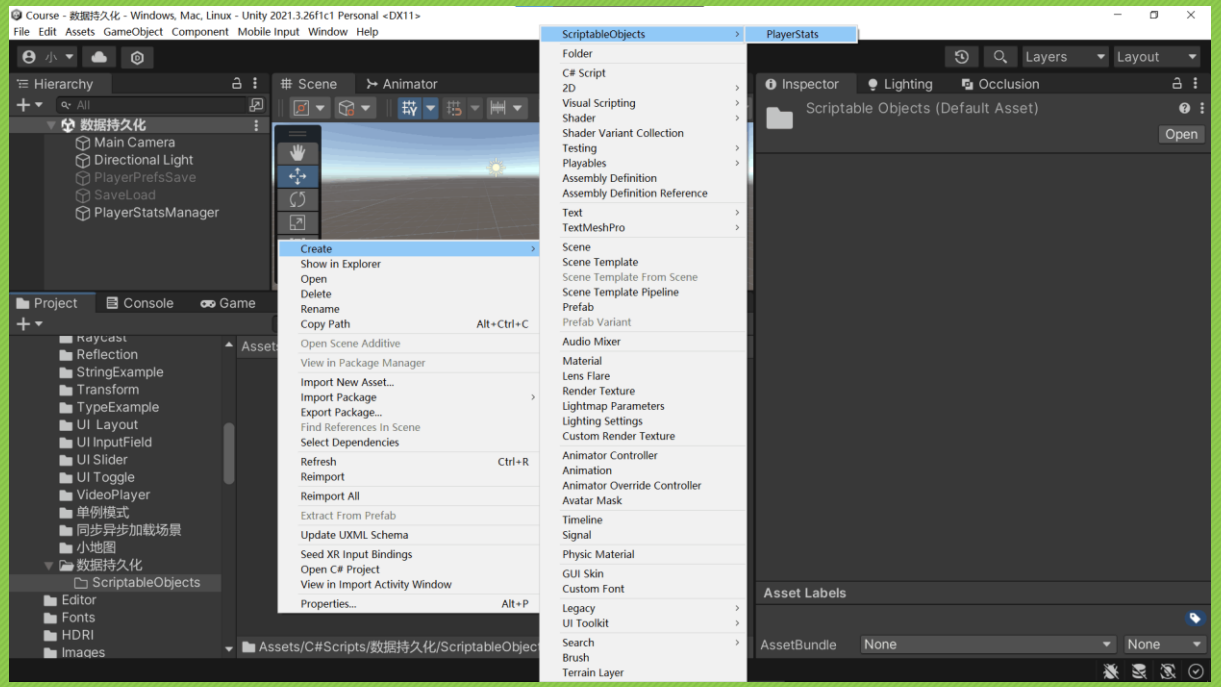
- 保存和加载**逻辑相对复杂**（手动实现在运行时保存为JSON或XML文件）。
- 只适用于静态数据，不适合保存玩家动态数据，无法处理动态的运行时数据持久化。

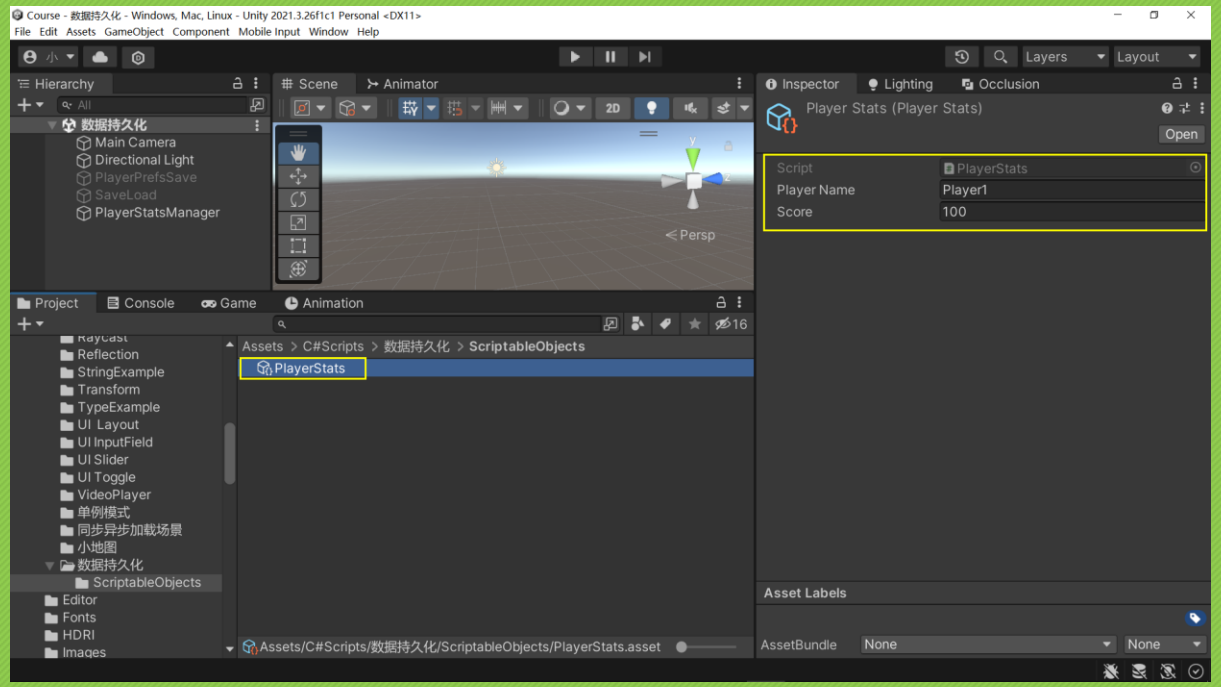
方法三：ScriptableObjects（具体实现）

```csharp
// 为ScriptableObject 类在 Unity 编辑器的Assets > Create 菜单中添加一个自定义选项
[CreateAssetMenu(fileName = "PlayerStats", menuName = "ScriptableObjects/PlayerStats", order = 1)]
public class PlayerStats : ScriptableObject
{
    // 玩家名称
    public string playerName = "DefaultPlayer";
    // 玩家分数
    public int score = 0;

    /// <summary>
    /// 增加分数
    /// </summary>
    public void AddScore(int amount)
    {
        score += amount;
    }

    /// <summary>
    /// 重置玩家数据
    /// </summary>
    public void ResetStats()
    {
        playerName = "DefaultPlayer";
        score = 0;
    }
}
```

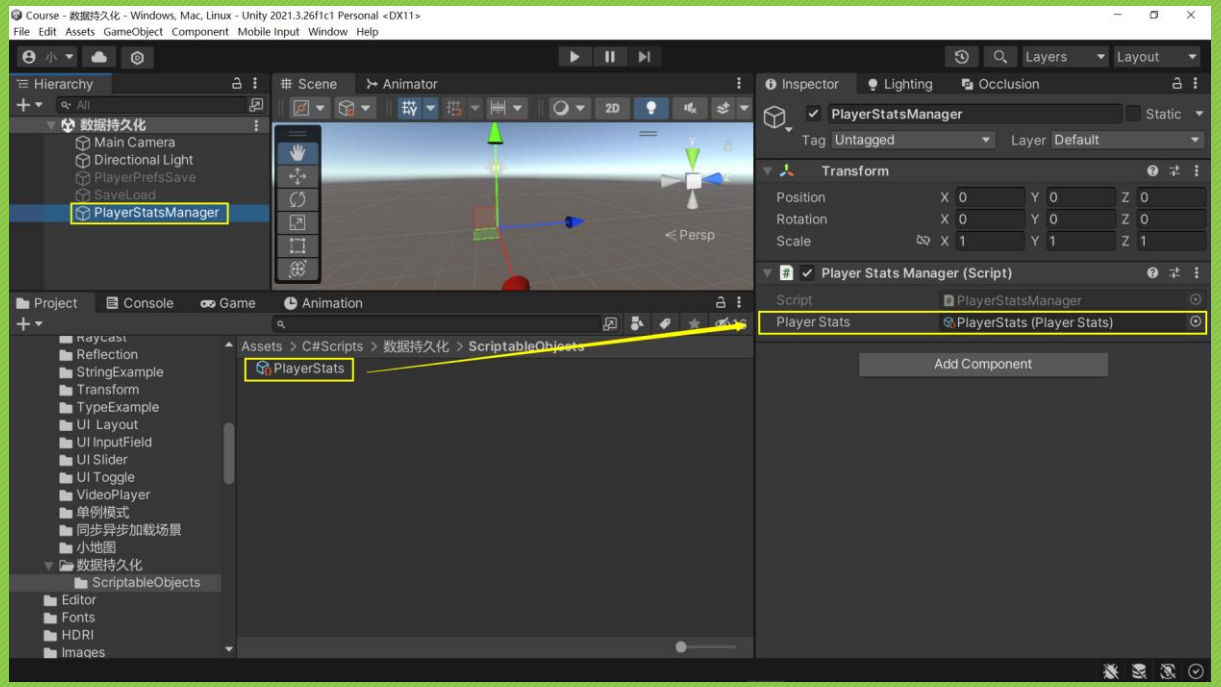
```csharp
public class PlayerStatsManager : MonoBehaviour
{
    public PlayerStats playerStats; // 指定的 ScriptableObject 实例

    private void Start()
    {
        // 显示初始状态
        Debug.Log($"玩家名称: {playerStats.playerName}, 初始分数: {playerStats.score}");
    }

    private void Update()
    {
        // 按键控制功能
        if (Input.GetKeyDown(KeyCode.A)) // 增加分数
        {
            playerStats.AddScore(10);
            Debug.Log($"玩家 {playerStats.playerName} 当前分数: {playerStats.score}");
        }

        if (Input.GetKeyDown(KeyCode.R)) // 重置数据
        {
            playerStats.ResetStats();
            Debug.Log($"玩家数据已重置, 当前名称: {playerStats.playerName}, 分数: {playerStats.score}");
        }
    }
}
```

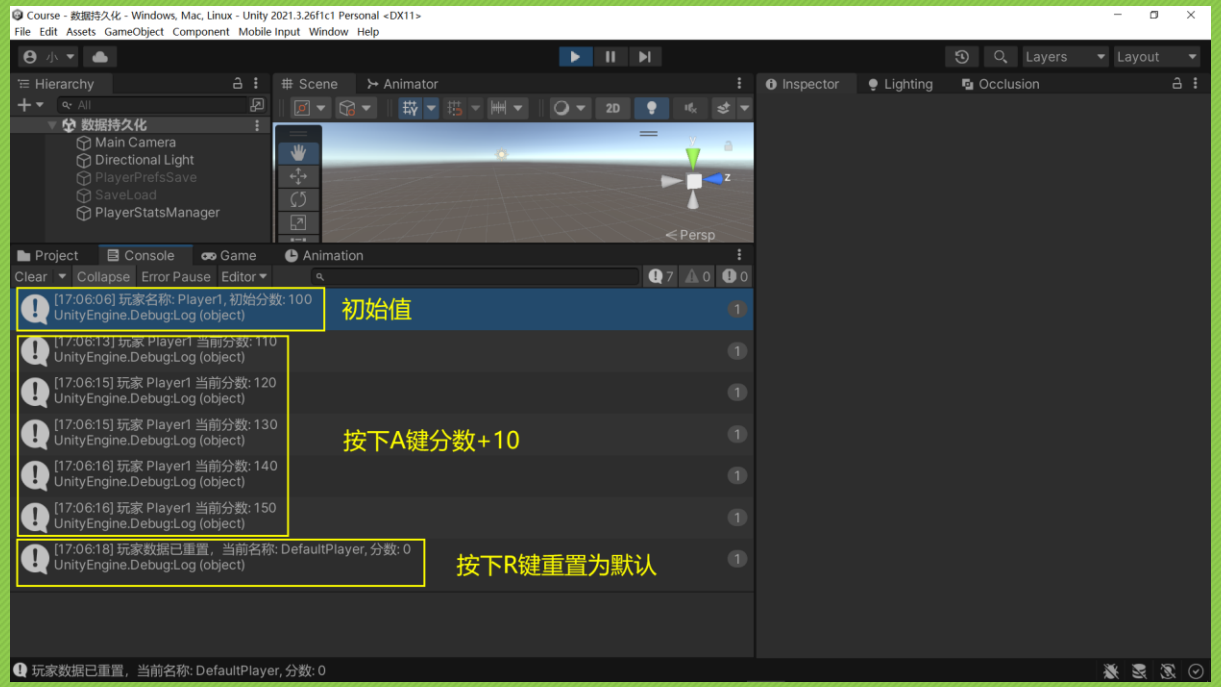
方法三：ScriptableObjects（具体实现1）





方法三：ScriptableObjects（具体实现2）





三种方法对比（总结）

方法	优点	缺点	适用场景
PlayerPrefs	简单易用，内置支持，跨平台，无需额外文件操作	数据类型单一，仅适合小型数据，容量有限	用户设置（音量、分辨率）、轻量级进度存储
JSON	灵活支持复杂数据结构，跨平台兼容性强，易扩展	文件操作复杂，不适合频繁动态数据更新	游戏存档，动态生成内容，网络数据存储
ScriptableObject	数据可视化，直观编辑，适合跨场景共享和静态配置	默认不可本地持久化，需额外处理保存逻辑	游戏配置（关卡设计、武器属性）、全局状态

简单需求（如存储音量、分辨率）：PlayerPrefs。

复杂动态数据（如游戏存档、多样化数据结构）：JSON。

静态配置数据（如关卡设计、装备参数）：ScriptableObject。

【Unity基础教程】重点知识汇总

（十五）

Unity实现数据持久化的三种常见方法