Unity 调用动态链接库方法研究

赵 鹏[1] 孔璐璐[1] 卢亚飞[1]
ZHAO Peng KONG Lulu LU Yafei

摘 要 针对 Unity 工业应用场景中动态链接库调用的需求，文章在分析不同类型动态链接库差异的基础上，系统梳理了 Unity 调用托管与非托管动态链接库的技术路径与实践案例；通过实验量化分析了 64 位操作系统下调用 32 位动态链接库的延时等性能瓶颈，并提出了有效的性能改进方案，为 Unity 与第三方硬件 / 软件的高效集成提供了技术参考。

关键词 Unity 引擎；动态链接库；托管代码；非托管代码

doi：10.3969/j.issn.1672-9528.2026.07.009

0 引言

Unity 是一款优秀的三维引擎，在游戏、交互式三维仿真等领域得到了广泛的应用，其具备强大的渲染能力，丰富的生态资源 [1-2]。但其在工业应用领域等对需要读取或控制硬件信息的场合，往往要通过动态链接库的方式调用相关软件的实时数据或仿真计算结果以驱动场景中渲染对象的状态更新。这时，需要通过 Unity 调用第三方软件的动态链接库 [3]。

动态链接库（DLL）是 Windows 系统中实现代码复用与功能模块化的核心组件，广泛应用于硬件驱动、算法封装、第三方工具集成等场景。Unity 作为当前主要三维仿真和游戏引擎，在许多场合需要通过动态链接库（DLL）调用外部程序数据以实现实时场景驱动、仿真计算等功能。而随着操作系统的不断变化，动态链接库的格式和调用方式也在不断变化，从一定程度上导致动态链接库调用失败，严重影响开发进度。本文从调用角度分析不同版本动态链接库的区别和调用方法，使其能够快速完成 Unity 与外部软件的交互过程，加快开发进度。

1 动态链接库分类

Windows 平台下，动态链接库分为 32 位和 64 位，还分

1. 陆军兵种大学郑州校区 河南郑州 450052

参考文献：

[1] 符昌昊 , 高荧荧 , 许秀英 . 无人机在农作物检测领域应用现状及展望 [J]. 现代化农业 ,2025(1):88-90.

[2] 陈丹 , 徐天枢 , 马园园 , 等 . 复杂城市低空物流无人机任务协同动态分配 [J/OL]. 航空学报 ,1-22[2026-06-22].https://link.cnki.net/urlid/11.1929.V.20251017.1512.016.

[3] 伍刚 . 无人机在公共安全领域的创新应用 [J]. 中国安防 , 2024(9): 19-23.

[4] 张延德 . 基于无人机倾斜摄影测量的露天煤矿灾害风险评估 [J]. 中国矿山工程 ,2025,54(5):80-86.

[5] 蒋文全 , 高豪云 , 郑佳秋 , 等 . 无人机在民用行业应用研究综述 [J]. 机电工程技术 ,2025,54(9):119-124.

[6] 张梦婷 , 张军 , 何承烽 . 元启发式算法研究综述 [J/OL]. 计算机工程与应用 ,1-16[2026-06-22].https://link.cnki.net/urlid/11.2127.TP.20250910.1253.002.

[7] 代进进 , 戢治洪 , 李季颖 , 等 . 基于改进遗传算法的多无人机协同搜索方法 [J/OL]. 指挥控制与仿真 ,1-7[2026-06-22]. https://link.cnki.net/urlid/32.1759.tj.20251113.1131.002.

[8] 卢秀丽 , 安娟华 , 冀松 , 等 . 基于粒子群优化的 WSN 多目标 3D 无人机运动规划 [J]. 传感器与微系统 , 2025, 44(11): 153-158.

[9] 胡佳伟 , 张佳伊 , 裴庆雨 . 一种基于改进蚁群算法的无人机路径规划方法 [J]. 无线电工程 ,2025,55(10):2105-2113.

[10] 郭志军 , 王远 , 王丁健 , 等 . 移动机器人反馈式步长萤火虫算法全覆盖路径规划研究 [J]. 重庆理工大学学报 (自然科学), 2025,39(4):148-156.

【作者简介】

刘晓波（1995—），男，山西大同人，博士研究生，讲师，研究方向：人工智能、深度学习、强化学习、优化算法和计算机视觉等。

刘海芳（1989—），女，山西晋中人，博士研究生，副教授、讲师，研究方向：混沌电路、随机数产生、图像处理等。

（收稿日期：2025-12-01 修回日期：2026-07-13）

为托管和非托管代码[4]。二者的调用流程存在明显差异，本文先从 Unity 调用动态链接库的视角，梳理 Windows 平台下动态链接库的类型。

1.1 32 位和 64 位动态链接库

早期操作系统和应用软件均为 32 位系统，最大只支持 4 GB 虚拟内存空间，其长整型变量 long 为 32 位，指针为 32 位。随着硬件的更新，64 位版本操作系统出现并逐步替代了 32 位操作系统，直到 Windows11 操作系统出现，微软彻底停止了 32 位操作系统的开发。相比 32 位系统，64 位系统其长整型变量 long 变为 64 位，指针也为 64 位。支持更大内存（Windows10 专业版支持 2 TB 内存），提供更高的性能。同时，32 位操作系统和应用软件的动态链接库也为 32 位，而 64 位操作系统和应用软件的动态链接库则是 64 位。虽然当前 Windows 操作系统仍支持 32 位应用软件的运行，但 64 位应用软件却无法调用 32 位动态链接库。

查看动态链接库是 32 位还是 64 位，除通过其运行环境判断外，还可以借助软件判断。比如，通过 Visual Studio 工具提供的 dumpbin 命令，可以判断 DLL 是否为 32 位。命令是 dumpbin/headers 加目标 DLL 路径和文件名。查看输出中“machine”字段：x86 对应 32 位，x64 对应 64 位。除此以外，还有 Dependency Walker PE Explorer 等软件也可以判断动态链接库的位数。

1.2 托管和非托管代码动态链接库

托管代码是 NET 框架特有的概念，依赖公共语言运行时（CLR）管理。由托管代码形成的动态链接库就是托管动态链接库。基于 NET Framew/.NET Core 开发，包含 MSIL 中间代码，运行时需要 NET CLR（公共语言运行时）执行解释，具有跨平台属性。托管代码开发效率高，依托 NET 框架类库，支持C＃等高级语言，无需手动管理内存（CLR 自动垃圾回收），减少内存泄漏。CLR 严格校验数据类型，能减少数组越界、空指针等低级错误，代码稳定性有所提升。但上述特性也带来额外性能开销，不适合实时性要求极高的场合，对系统底层 API 和硬件资源的直接操作能力较弱，且运行时必须安装。NET Framework 运行时库，否则无法运行。

而非托管动态链接库基于原生代码（C/C++ 等）开发，包含 CPU 可直接执行的机器码，在 Framework .net 开发环境出现前，代码都直接由操作系统执行，需要手动处理内存管理、类型校验等操作，比如 Visual Basic6、Visual C++6 编写的代码，及早期 C 编译器生成的代码，均属于非托管代码范畴。非托管代码运行时直接由 Windows 操作系统加载，无需 CLR 依赖，具有执行效率高，无需额外环境依赖，底层控制能力强的特点。但开发门槛较高，需手动管理内存，容易出现内容泄漏等问题，调试难度大、代码容错率低，需要开发者自行保证安全性[5]。

判断动态链接库是托管或非托管，同样可借助 dumpbin 命令。其格式为：

dumpbin /headers /clrheader 你的文件 dll | findstr "CLR"

* 如果输出中包含 clr Header，则是一个托管 DLL（或混合模式程序集）。

* 如果输出显示 clr Header 不存在或找不到，则它是一个非托管 DLL。

例如，运行 dumpbin /headers C:\Windows\Microsoft.NET\Framework\v4.0.30319\mscorlib.dll | findstr "CLR"

会输出：0 CLR Header 大小，证明 mscorlib.dll 是托管的。

运行 dumpbin /headers C:\Windows\System32\kernel32.dll | findstr "CLR"

没有相关输出，证明是 kernel32.dll 非托管的。

2 调用方法

由于动态链接库具有上述不同，调用方式也有较大的区别。

2.1 调用 32 位或 64 位动态链接库

CPU 架构的 32 位与 64 位环境在内存寻址、寄存器宽度、数据类型对齐方式上存在根本差异：32 位系统最大支持 4 GB 内存寻址，64 位系统则支持更大内存空间[6]。Unity 自 2018 版本后默认采用 64 位编辑器及编译环境，但在军事装备模拟、工业控制仿真等专业开发中，大量仍在使用的陈旧系统及专用硬件配套的 SDK 仍以 32 位 DLL 形式提供。Unity 编辑器及生成的可执行文件若为 64 位，当其调用 32 位动态链接库时，会试图以 64 位方式加载 32 位 DLL，导致内存寻址冲突与指令集不兼容，最终触发加载失败；若将程序编译为 32 位版本，虽然编译后的可执行程序可调用 32 位动态链接库，但在开发过程中无法进行调试，会给开发过程带来诸多不便。

在这一情况下，只能改用早期 32 位版本的 Unity 编辑器进行开发，但会影响高版本 Unity 的先进特性。如果一定要在 64 位 Unity 版本中调用 32 位动态链接库，需要用其他软件编写一个 32 位的外壳程序，由其加载 32 位动态链接库，Unity 和这个外壳程序通过命名管道、Socket、内存映射文件等手段实现数据交换。

2.2 调用托管和非托管动态链接库

无论是 32 位动态链接库还是 64 动态链接库，都有可能是托管代码编写或者是非托管代码编写，其调用方式差别较大。

2.2.1 调用托管动态链接库

Unity 的脚本层基于 NET 框架（Mono 或 IL2CPP），采用托管代码（Managed Code）运行环境。所以其调用托管 DLL 比较简单，只要将 DLL 直接放入 Unity 工程的 Asset/

Plugins 文件夹即可，无需额外配置，Unity 会将其识别为托管插件。在调用时，直接在 Unity 的 C# 脚本开头引用 DLL 的命名空间，即可在后续调用动态链接库中的函数了。

例如：

在 Unity 的脚本开头通过如下语句，引入动态链接库的命名空间

using MathUtility; // 引用托管 DLL 的命名空间

在脚本里即可直接调用动态链接库里的函数。如果是静态类方法，无需实例化。比如调用动态链接库中的静态类 Calculator 的 Add 函数。

int sum = Calculator.Add(5, 3);

如果是动态类方法，需要先实例化。比如调用动态链接库中的动态类 DataProcessor 的 GetVersion 函数。

DataProcessor processor = new DataProcessor("1.0.0");

string version = processor.GetVersion();

Unity 引擎对托管 DLL 的支持需满足以下条件：

（1）框架版本要兼容。Unity 不同版本对 NET 框架的支持存在差异，Unity 2019—2020 主要支持 NET Framework 4.6、NET Standard 2.0；Unity 2021 及以上则新增对 NET 6/7/8 的支持，同时兼容 NET Standard 2.1；托管 DLL 的开发框架版本需与 Unity 项目的 NET 设置一致，否则会出现“版本不兼容”“类型未找到”等错误。

（2）平台架构匹配。托管 DLL 的编译架构（如 x86、x64、ARM）需与 Unity 目标平台的架构一致。例如，Windows 平台下 64 位 Unity 项目需调用 x64 架构的托管 DLL，Android 平台需调用 ARM 架构的 DLL。

（3）程序集导入规则。Unity 默认从 Assets/Plugins 文件夹加载托管 DLL，不同平台的 DLL 需放置在对应子文件夹（如 Plugins/Windows、Plugins/Android），并在 Inspector 面板中配置目标平台与 CPU 架构，确保 Unity 在打包时正确识别并嵌入 DLL。

（4）如果 DLL 中包含了指针等“不安全代码”，需要在 Unity 的 Edit>Project Settings>Player 设置中勾选 Allow unsafe Code 选项。如果在 Unity 中调试 DLL 的源代码，需要将 DLL 编译时生成的 pdb 文件与 DLL 文件一同放入 Unity 项目。

2.3 调用非托管 DLL

如果 Unity 调用的动态链接库属于非托管代码（Unmanaged Code），则两者的交互需通过 NET 提供的交互技术实现，核心包括 P/Invoke 与 COM Interop 两种方式。由于动态链接库提供 COM 接口的不多，本文主要介绍 P/Invoke 技术。

2.3.1 P/Invoke 技术

P/Invoke（platform invocation services）是 NET 框架提供的跨平台调用非托管函数的技术，通过“平台调用服务”将托管代码的函数调用请求转换为非托管代码的函数调用，具体流程如下：

（1）函数声明：在 C# 中使用 [DllImport] 特性声明非托管 DLL 中的函数，指定 DLL 文件名、函数名、调用约定（如 StdCall、Cdecl）等参数；

（2）数据封送：将 C# 中的托管数据类型（如 string、int []）转换为非托管代码支持的数据类型（如 char*、int*），反之亦然；

（3）函数调用：通过平台调用服务调用非托管 DLL 中的函数，执行指定功能；

（4）结果返回：将非托管函数的返回值与输出参数转换为托管数据类型，返回给 C# 脚本。

P/Invoke 技术的优势在于无需修改 DLL 源码，即可实现 Unity 与现有 DLL 的交互，适用于 Windows、Android 等多平台；缺点是数据封送过程可能产生性能开销，且需手动管理非托管内存，易引发内存泄漏。

2.3.2 调用方法

Unity 的 C# 脚本对原生 DLL 的 P/Invoke 调用核心是 System.Runtime.InteropServices 命名空间下的 DllImport 特性 [7]。DllImport 通过指定 DLL 路径、函数入口点（EntryPoint）、字符集（CharSet）等参数，建立 C# 函数与 DLL 导出函数的映射关系。调用流程如下：

（1）C# runtime 加载目标 DLL 到当前进程地址空间；

（2）完成 C# 数据类型到原生 DLL 数据类型的 Marshal 转换；

（3）调用 DLL 导出函数并获取返回值；

（4）释放 Marshal 资源，完成调用流程。

（5）需注意，P/Invoke 要求调用方进程与 DLL 位数一致，即 32 位进程只能加载 32 位 DLL，64 位进程只能加载 64 位 DLL，否则会抛出 BadImageFormatException 异常 [7]。

Unity 中 csharp 脚本示例代码如下：

```
using System;
using System.Runtime.InteropServices;
using UnityEngine;
public class DLLCaller : MonoBehaviour
{
  [DllImport("My32BitDLL.dll", CallingConvention = CallingConvention.Cdecl, CharSet = CharSet.Ansi)]; // 声明 32 位 DLL 中的导出函数（需与 DLL 实际函数名、参数类型一致）
   private static extern int Add(int a, int b); // 示例：两数相加函数
   void Start()
   {
     try
     {
       int result = Add(2, 3);
         Debug.Log("32 位 DLL 调用成功，结果：" + re-
```

```
sult);
        }
        catch (Exception e)
        {
            Debug.LogError("DLL 调用失败：" + e.Message);
        }
    }
}
```

3 实验分析

由于 Unity 早在 2017.1 版本之后即停止了对 32 位系统的支持，2017 年以后的系统编辑器和运行都要求基于 64 位操作系统，Unity 调用托管代码相对较为简单，所以选择目前开发过程中相对问题比较多的在 64 位操作系统中调用 32 位非托管动态链接库（采用 DLLProxy32.exe 外壳程序加载动态链接库）的情况进行实验比对。

3.1 实验环境

实验环境如表 1 所示。

表 1 实验环境

设备 / 软件	规格 / 版本
操作系统	Windows 10 专业版 64 位
Unity 引擎	2022.3.10 LTS
开发工具	Visual Studio 2022 17.5.0
测试 32 位 DLL	Test32DLL
代理进程	DLLProxy32.exe（32 位）
性能测试工具	Unity Profiler、Wireshark

3.2 实验内容与结果

在 64 位 Unity 编辑器中运行 ProxyDLLCaller.cs，测试不同数据量下的通信延迟（从 Unity 发送请求到接收响应的时间），每组测试重复 100 次，结果如表 2 所示。

表 2 中间层代理方案性能测试结果

数据量 / 字节	平均延迟 /ms	最大延迟 /ms	最小延迟 /ms	标准差 /ms
16（Add 函数）	12.3	28.5	8.1	3.2
1 024（字符串）	15.7	32.1	9.8	4.5
10 240（数组）	23.9	41.2	14.3	5.8

结果表明，中间层代理方案的平均延迟均低于 25 ms，满足工业可视化、中小型游戏等场景的实时性需求（通常要求延迟 < 100 ms），解决了 64 位 Unity 调用 32 位 DLL 的兼容性问题，延迟在可接受范围内，适用于需 64 位 Unity 特性的场景。

3.3 优化方向

为进一步减少延时，可将 Socket 通信替换为共享内存（如 MemoryMappedFile），可将延迟降低 50% 以上，适用于大数据量传输场景；还可采用数据序列化优化方法，使用 Protobuf 替代 JSON / 字符串序列化，减少数据传输量，提升通信效率。为降低操作复杂性，可将代理进程启动自动化，在 Unity 脚本中添加代理进程的自动启动、监测与重启逻辑，降低工程部署复杂度。

4 结论

本文针对 Unity 调用动态链接库的问题，较为全面地分析了几种动态链接库的不同，给出了不同动态链接库的调用方法。尤其分析测试了 64 位系统下调用 32 位动态链接库时，中间层代理方案的平均通信延迟，并给出了改进建议。为 Unity 项目集成 legacy 32 位 DLL 提供了技术支撑。后续研究可聚焦于跨平台扩展与通信性能优化，进一步提升方案的适用性与效率。

参考文献：

[1] 田欣 , 何毅 , 王水涛 .Unity 实战驱动的 C# 程序设计教学革新 [J]. 计算机教育 ,2025(2):150-154.

[2] 邬远哲 , 张静 , 刘振东 , 等 . 基于虚拟仪器技术的船舶电场仿真与信号生成研究 [J]. 舰船科学技术 ,2024,46(15):125-129.

[3] Unity Technologies. Unity Manual[EB/OL]. [2026-06-22]. https://docs.unity3d.com/Manual/index.html.

[4] 李伟 .C# 调用 VC++ 动态链接库的研究 [J]. 测控技术 , 2013, 32(5): 105-108.

[5] Garcia M, Lopez V, Rodriguez J. Performance analysis of managed vs. native plugins in Unity for real-time simulation[J]. Simulation Modelling Practice and Theory, 2019, 94: 148-162.

[6] Intel Corporation. Intel 64 and IA-32 architectures software developer's manual [EB/OL]. (2026-04-06)[2026-06-22]. https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html.

[7]Microsoft Corporation.NET documentation: platform invoke (P/Invoke)[EB/OL].(2026-04-25)[2026-06-22].https://learn.microsoft.com/en-us/dotnet/standard/native-interop/pinvoke.

【作者简介】

赵鹏（1976—），男，河南偃师人，博士，研究方向：计算机技术。

（收稿日期：2025-12-09 修回日期：2026-07-14）