Debugx 用户手册(UserManual)

简介

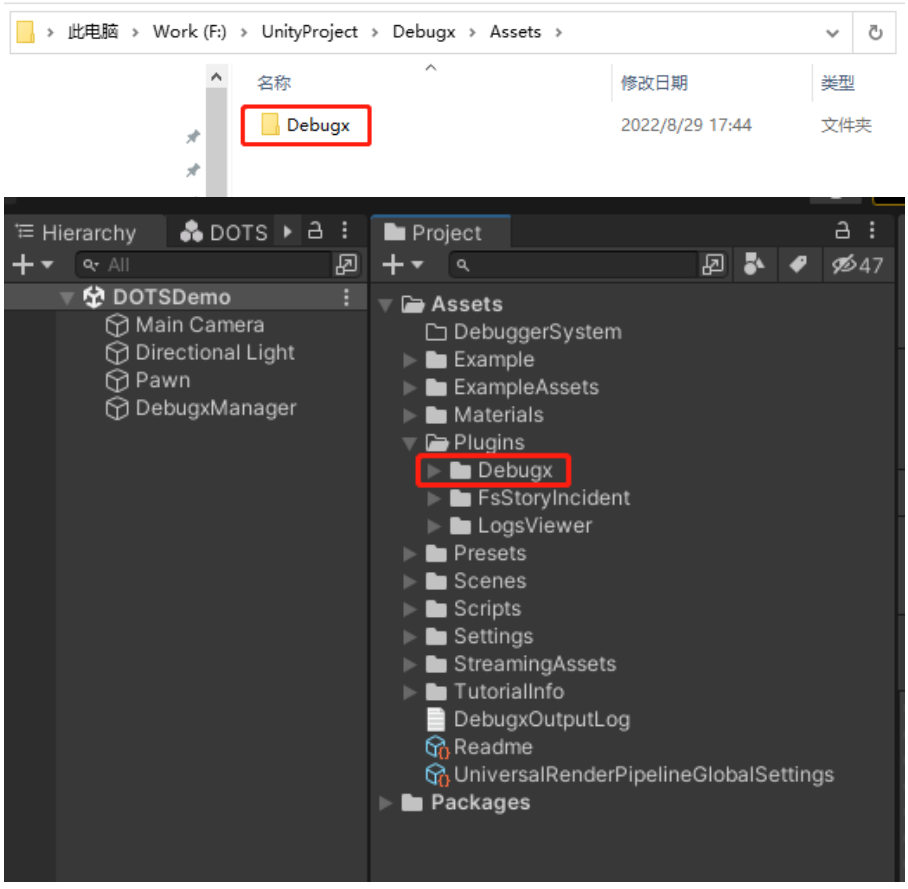
Debugx 插件用于按成员管理我们的 Log 打印。之后输出 Log 文件存储到本地方便查看。
在多人开发项目时，所有人都使用 UnityEngine.Debug.Log()会导致 Log 难以管理和区分。我们在测试我们的功能时，并不想被其他人的 Log 影响。
通过 DebugxManager 类，我们可以方便的管理所有的 Debugx 打印的 Log。

开始使用

跟随教程，快速上手 Debugx 插件。

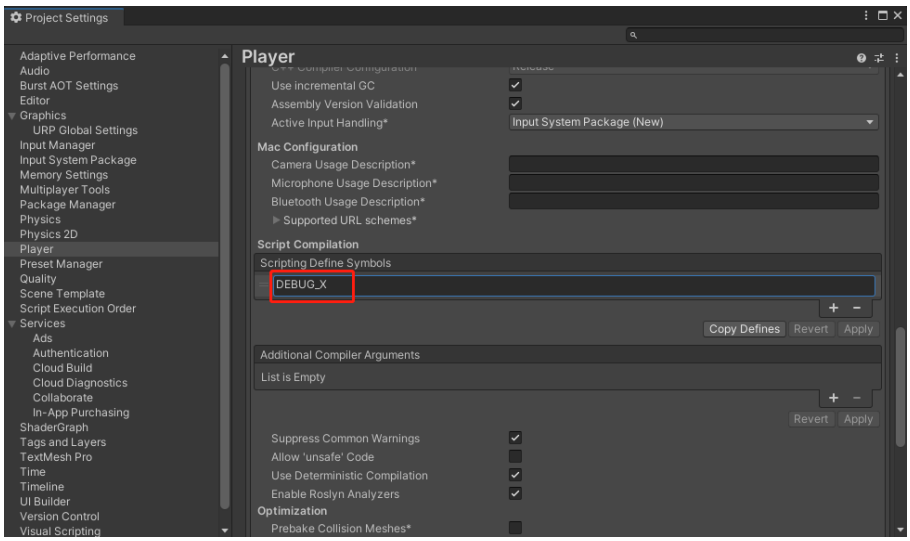
添加插件到你的项目

将项目中 Assets 下的文件夹拷贝到你的项目中。并等待编译完成。



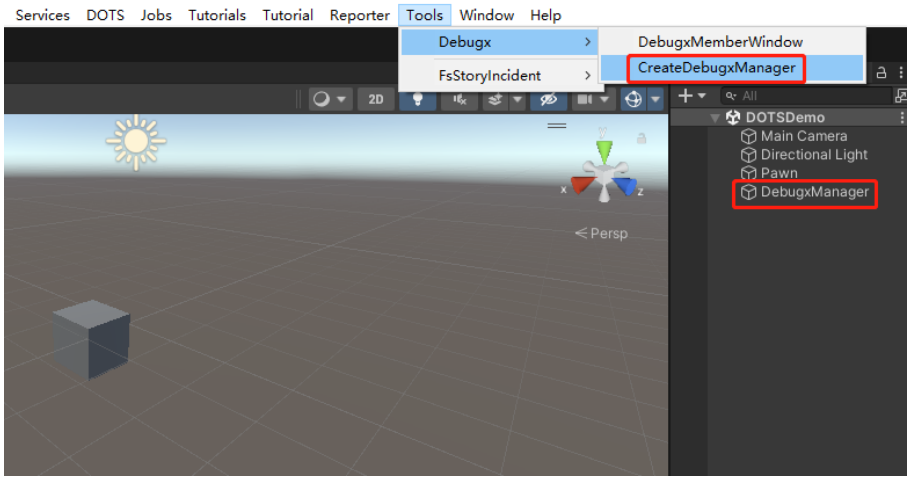
添加宏到你的项目

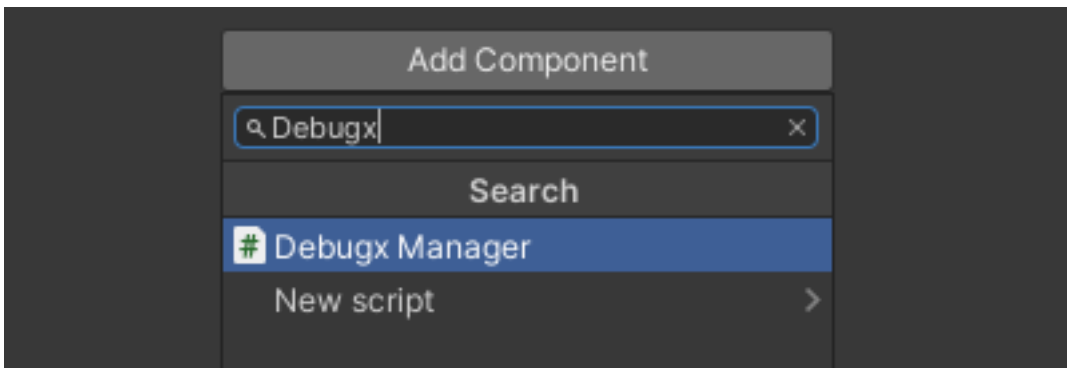
我们必须在项目中添加宏 DEBUG_X 才能开启打印功能。在项目打运行包时，我们可以去除宏 DEBUG_X 来快速的屏蔽所有的 Log 打印。



创建 DebugxManager

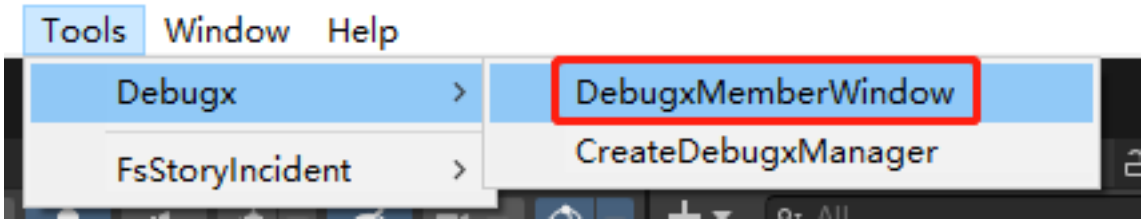
在场景中，通过菜单快速添加一个 DebugxManager，或者在你项目的启动预制体上添加 DebugxManager 脚本。



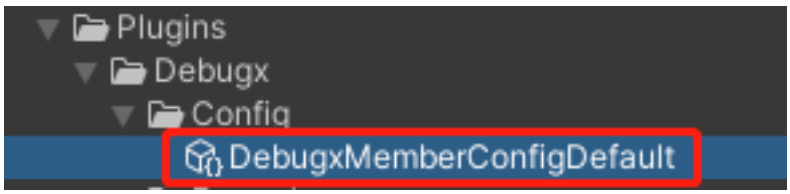


配置调试成员

配置参与调试的成员信息。我们可以使用编辑窗口进行编辑。你也可以直接修改.asset 可编辑资源。



初次打开 Window 时会自动创建一个默认的调试成员配置文件。



调试成员信息编辑器

我们有两个默认的成员，一个普通成员，一个高级成员。在成员信息列表，我们可以添加和删除我们自己的调试成员。

我们可以按自己的需求配置成员的信息，比如签名以及是否打印签名。

在字段上悬停鼠标，你可以看到我写的注释。

打印 Log

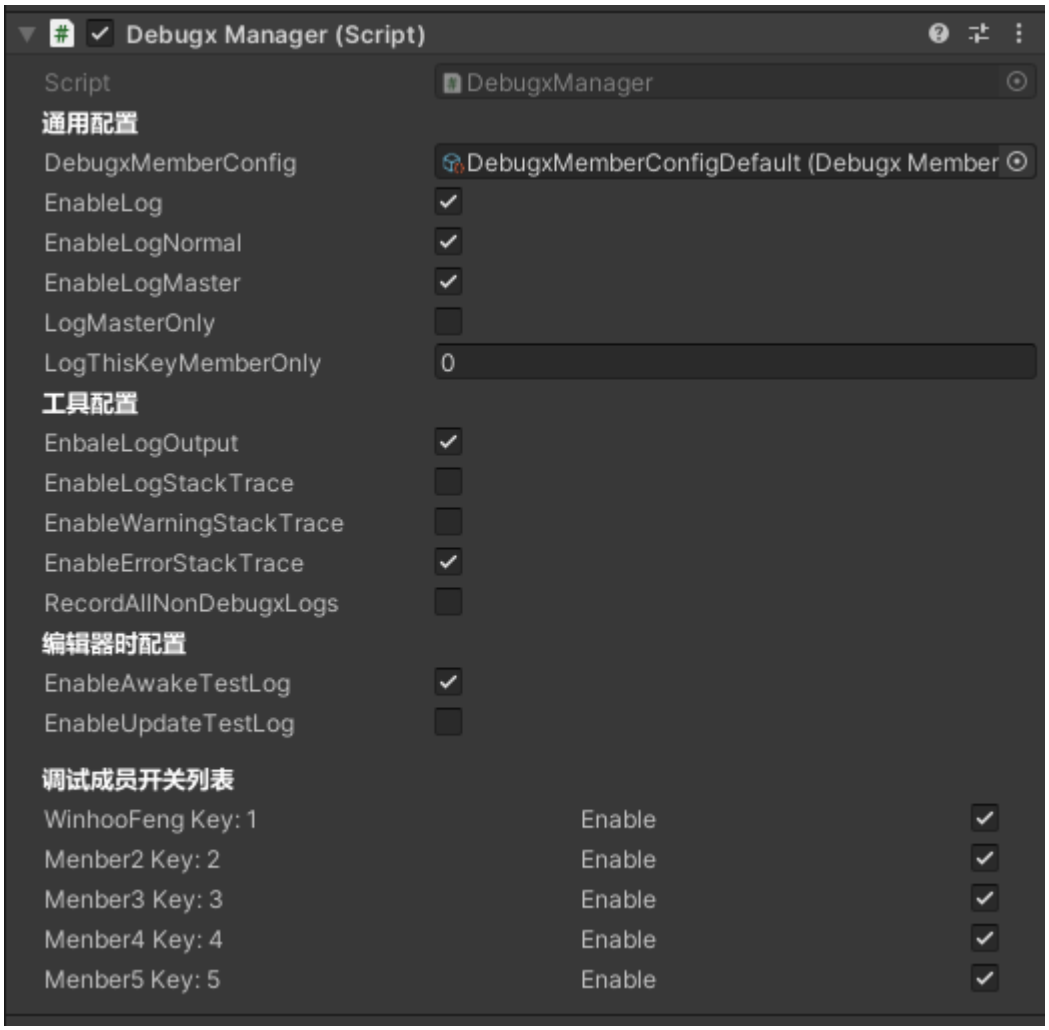
现在，我们可以开始打印我们的 Log 了。直接调用 Debugx 类（DebugxBurst 类在稍后解释）的静态方法来打印我们的 Log。
在 DebugxManager 类中，我已经调用了一些打印方法，用于快速的确认功能是否良好运行。你可以查看作为实例。

打印方法介绍

- Debugx.Log(key, message)
    - Log 系列方法是我们最常使用的方法，我们需要传入 Key 和打印内容。Key 是我们在调试成员配置中配置的成员所分配的 Key。每个成员需要记住和使用自己的 Key。
- Debugx.LogNom(message)
    - LogNom 系列方法是 Normal 普通成员使用的 Log 打印方法。一般成员不应当使用，否则容易混淆使用者。
- Debugx.LogMst(message)
    - LogMst 系列方法是 Master 高级成员使用的 Log 打印方法。除了主程，一般人都不应该直接使用此系列方法。
- Debugx.LogAdm(message)
    - LogAdm 系列方法是由 Debugx 插件的开发者使用的！任何人都不应当使用此方法，因为此方法打印的 Log 并不能通过 DebugxManager 来进行开关。但他还是受到宏 DEBUG_X 的影响。

DebugxManager

DebugxManager 单例用于在 U3D 中管理我们的 Debugx 打印的 Log。在脚本中开放了很多开关，通过改变这些开关，我们来控制允许哪些调试成员打印 Log（支持在运行时改变开关）。



通用配置

成员配置文件

- debugxMemberConfig
    - 目前配置到管理器的配置资源文件。在运行时初始化到 Debugx 类中。

成员 Log 开关

- enableLog
    - Log 总开关
- enableLogNormal

    - 普通成员开关
- enableLogMaster
    - 高级成员开关
- enableLogMember
    - 成员开关
- logMasterOnly
    - 仅打印高级成员
- logThisKeyMemberOnly
    - 仅打印此 Key 的成员。在 logMasterOnly==false 时才能设置，否则固定为-2 高级成员的 Key。

工具配置

- EnableLogOutput
    - 输出 Log 到本地文件，在 Editor 时文件会存储在项目的 Logs 文件夹下。在运行包时将会按平台存储在不同的位置。具体的位置由 Application.consoleLogPath 接口提供。在 PC 的运行包时一般会输出到 C:\Users\Winhoo\AppData\Local\Unity\Editor 位置。
- enableLogStackTrace
    - 打开 Log 的堆栈跟踪信息
- enableWarningStackTrace
    - 打开 Warning 的堆栈跟踪信息
- enableErrorStackTrace
    - 打开 Error 的堆栈跟踪信息
- recordAllNonDebugxLogs
    - 开启后，输出的本地 Log 文件也会记录所有非 Debugx 打印的其他 Log 信息。

编辑器时配置

此配置仅在编辑器 UNITY_EDITOR 时可用。在编辑器时，我们提供一些功能用于快速的调试或者编辑。

- EnableAwakeTestLog
    - 在 Awake 中打印测试 Log
- EnableUpdateTestLog
    - 在 Update 中打印测试 Log

DebugxBurst（为 DOTS 的 Burst 多线程提供的功能）

DebugxBurst 类主要用于 DOTS 的 Burst 多线程中进行 Log 打印。其中的 Log 方法基本和 Debux 中的一致，最终会调用到 Debugx 中的打印方法。必须在 Entities.ForEach().WithoutBurst().Run()时才能打印，否则 Log 会被排除。

DebugxBurst 的 Log 方法都添加了[BurstDiscard]标记，用于在 Burst 多线程时被排除。直接使用 Debugx 的 Log 方法会导致编译报错。

LogInBurst

我们还提供了 LogInBurst 系列方法允许在 Entities.ForEach().Schedule()时使用，但是此方法不支持任何调试成员信息。我们只能打印简单的 string。一般不推荐使用。

Burst 中的限制

一些受到限制的代码，一些功能不支持，在编译时会直接报红。

在 DOTS 的 Burst 多线程中，不能使用任何引用类型，string 只能直接传递，使用 String.Format 时不能传入 string 类型。

UnityEngine.Debug.unityLogger()不能使用。只能直接使用 UnityEngine.Debug.Log 等方法，这类方法应该是由引擎开封这做过特殊处理，从而能够直接传参 object 类型。

使用外部的值时，值必须是只读的。

in ref out 都不支持。

所以使用[BurstDiscard]特性用于在多线程时直接排除此方法，添加此宏能够使一些被限制的代码在编译时不报错，但必须在 Entities.ForEach().WithoutBurst().Run()才能工作;,因为所有[BurstDiscard]特性的方法在多线程中都不工作。

使用 LogInBurst()方法可以直接在 Burst 多线程中打印（但不支持任何成员配置信息），效果和 UnityEngine.Debug.Log()是一样的，这种方法应该坐过处理，所以直接传参 object 也不会报错