Debugx 用户手册(UserManual)

简介

Github：https://github.com/WinhooF/Debugx

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。
我们仅需要添加宏"DEBUG_X"到我们的项目，之后进行简单的配置，就可以开始使用 Debugx 的功能了。
Debugx 在 ProjectSettings 和 Preferences 中分别提供了配置界面，ProjectSettings 中的用于配置到项目，Preferences 中的用户配置仅会影响你个人，不会对项目和其他人造成影响。
DebugxConsole 用于在项目运行时操作打印开关等内容。

开始使用

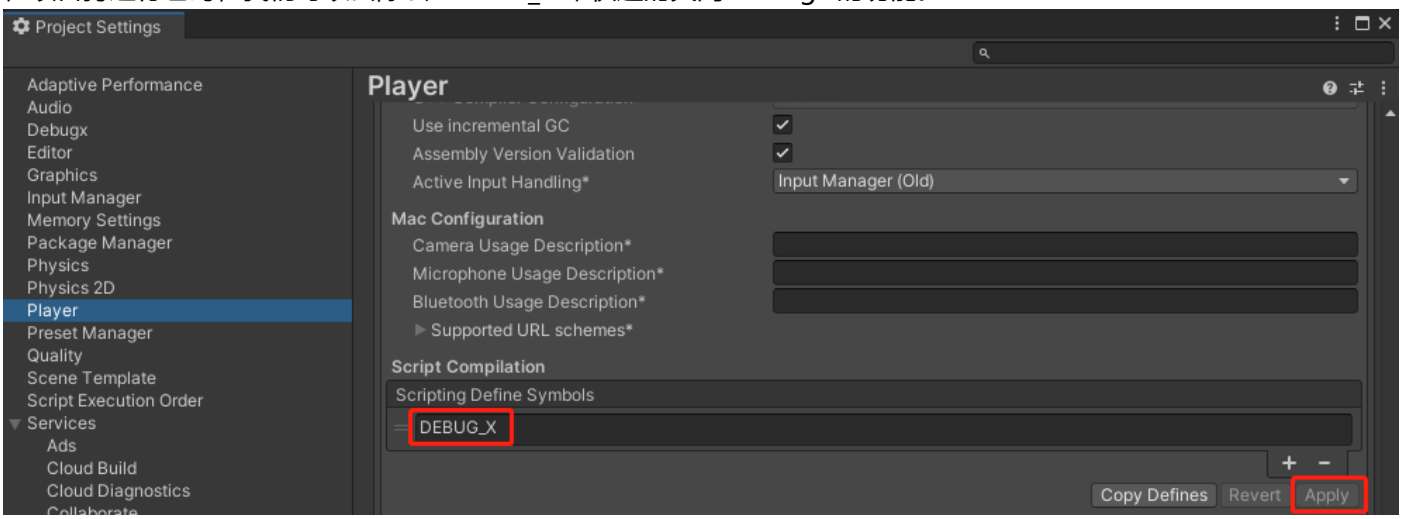
跟随教程，快速上手 Debugx 插件。

添加插件到你的项目

下载 Releases 包，使用.unitypackage 包安装 Debugx 插件到你的项目。
或者直接下载 Github 项目中的 Debugx 文件夹并放入你的项目中。

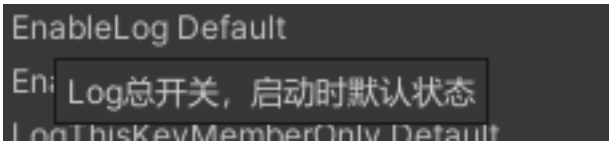
添加宏到你的项目

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 来快速的关闭 Debugx 的功能。



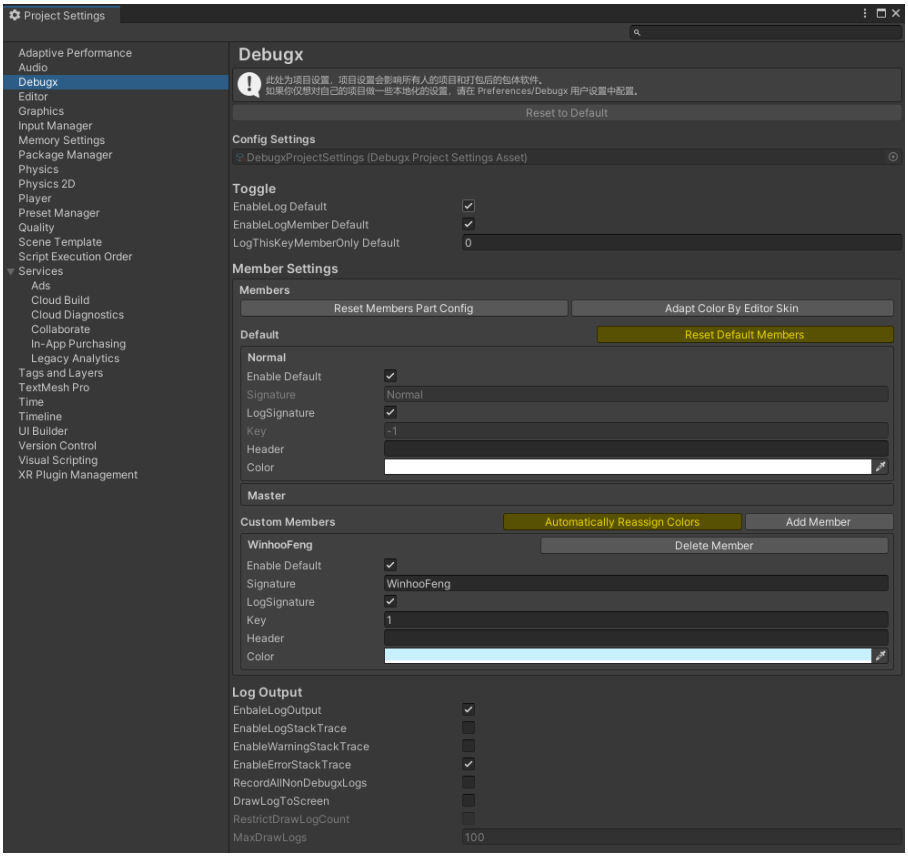
Debugx 配置

在开始前，我们需要先知道，鼠标悬停在字段上，会出现 tooltip 提示，这会更好的帮助你上手 Debugx。
所以我不会对每个条目进行介绍，因为你可以自己查看 tooltip。



ProjectSettings 项目设置

在 Editor>ProjectSettings>Debugx 中打开 Debugx 项目设置界面。
项目设置会对所有人的项目都产生影响，也会对打包后的项目产生作用。



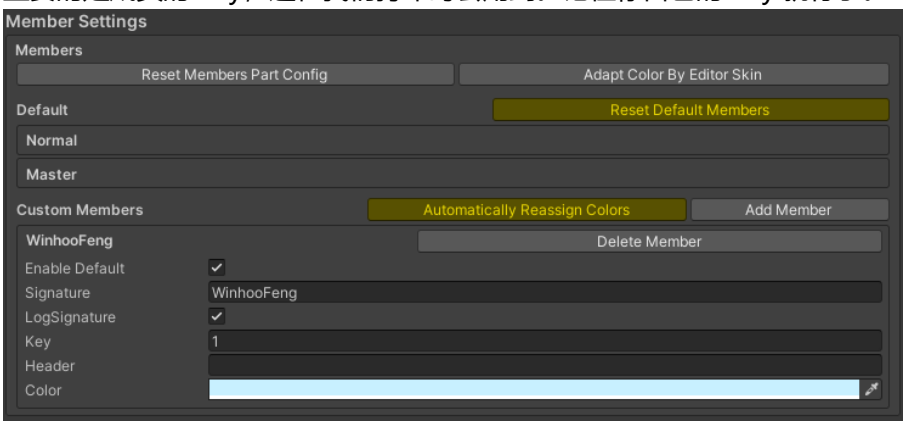
Toggle 开关

这里是一些开关设置。总控开关会在这显示，而调试成员可以在成员信息中单独设置开关。

MemberSettings 调试成员设置

成员设置用于配置调试成员。这里有一些默认的成员，他们不能被删除，仅能进行有限的编辑。

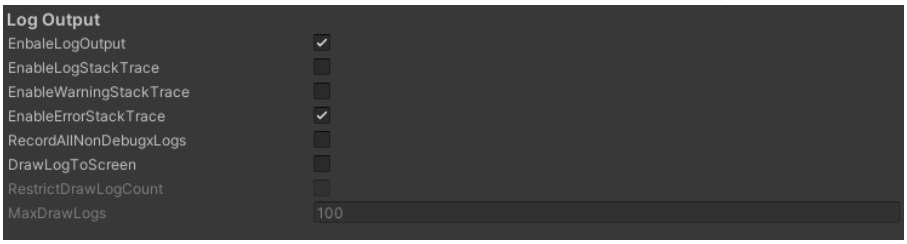
我们可以在自定义成员中添加你的专用成员配置，按项目的使用者去区分。我们可以设置开关，签名，颜色等内容。最重要的是成员的 Key，这在我们打印时会用到。记住你自己的 Key 就行了。



LogOutput

日志输出功能会在每次项目开始运行时开始记录，在项目停止运行时结束记录并输出到本地。在编辑器时，log 本地文件会输出到你项目根目录的 Logs 文件夹下。

在打包时，根据不同的平台，log 本地文件会存储到不同的目录中。PC 平台一般在 C:\Users\UserName\AppData\LocalLow\DefaultCompany\ProjectName 目录下。

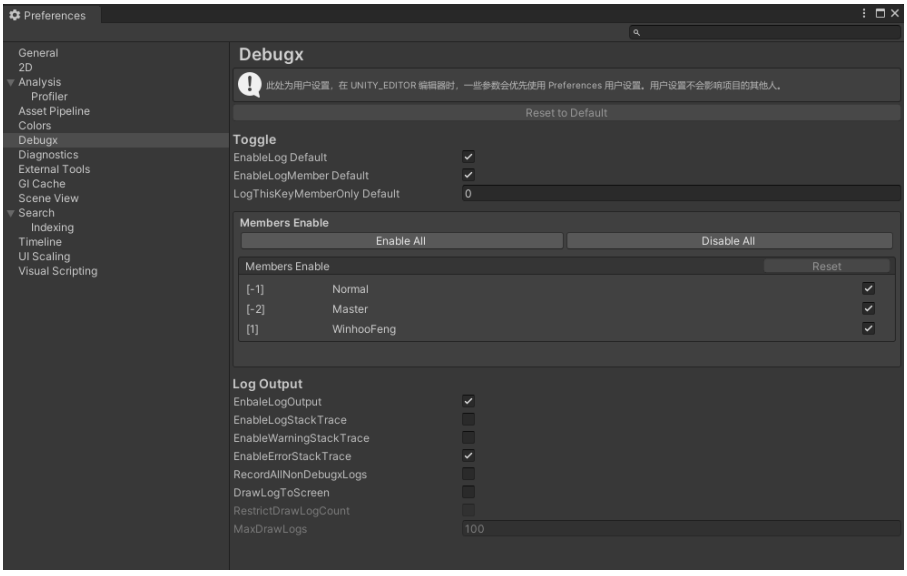


Preferences 用户偏好设置

在 Editor>Preferences>Debugx 中打开 Debugx 用户偏好设置界面。

用户偏好设置仅会对你个人本地的项目产生影响，并不会影响其他人的项目。也不会对打包产生作用。

用户设置内容基本上和 ProjectSettings 项目设置相同，主要是为不同的开发者在自己本地按个人需求进行配置，比如每个人在自己的项目上，一般只会打开自己调试成员开关。因为我们不想被其他人的调试打印影响。



在代码中打印 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。

```
Debugx.LogNom("LogNom Print Test");
Debugx.LogMst("LogMst Print Test");
Debugx.Log(1, "Key 1 Print Test");
```

打印方法

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。也可以让所有成员在打印通用的报错或者警告时使用 LogNom，保证一些关键信息总是保持打印。

Debugx.LogMst(message)

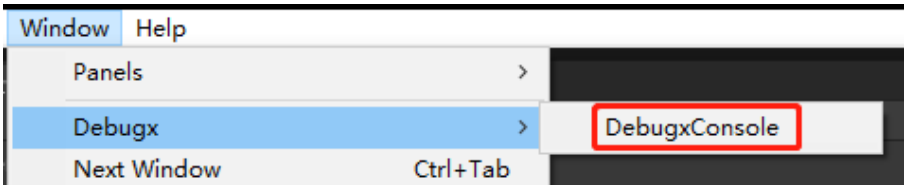
LogMst 系列方法是 Master 高级成员使用的 Log 打印方法。除了主程，一般人都不应该直接使用此系列方法。

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。

DebugxConsole

debugx 控制台主要用于在项目运行时对 Debugx 功能进行一些开关操作。在 Window>Debugx>DebugxConsole 中打开窗口。
为了方便，我们可以将它和 Game 页签放在一起。

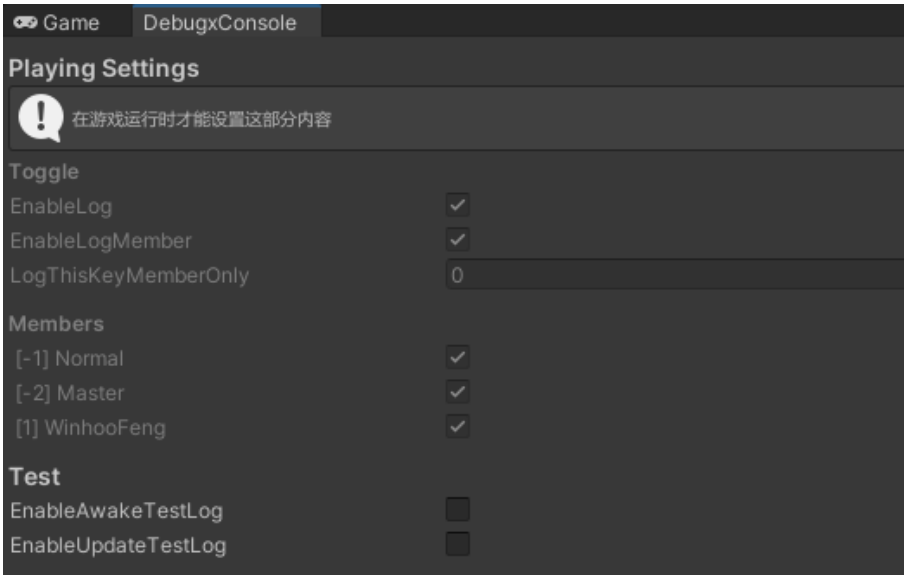


PlayingSettings

项目运行时设置的内容基本上和 ProjectSetting 中的一样，只是这是允许在运行时设置的。

Test

测试用功能。提供了一些方便的测试用功能开关，用来确认 Debugx 功能是否正常的运作了。



DebugxManager

DebugxManager 在游戏运行时自动创建，我们一般不用去管他。他的主要工作是对 LogOutput 进行操作。
只有在 DEBUG_X 宏添加到项目时，DebugxManager 才会自动创建。

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。因为[BurstDiscard]特性，Debugx 源码中的字典，列表和数组都不会导致 Burst 的报错。

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。