ZLog日志系统

版本日志

编号	版本号	日期	更新说明	修订人
1	1.0.0	2022/2/15	第一版	陈伟淦
2	1.0.1	2022/2/18	添加空日志输出者（NullLogger），添加默认初始化的配置	陈伟淦
3	1.0.2	2022/2/19	添加按输出等级输出日志到对应文件的logger（SplitFilesLogger），并设置其为默认初始化的配置	陈伟淦

文件目录结构

```
|
├─ZLogExample                    // 示例
|    SampleScene.unity
|    TestLog.cs
|
└─ZLog                           // 日志核心代码
```

```
Options.cs                  // 日志的配置文件，初始化日志的等级。 如果使用ET框架，去除这个文件，使用ET自身的Options
ILog.cs                     // 日志接口
Log.cs                     // 日志管理
NullLogger.cs               // 日志输出者 不输出日志
FileLogger.cs              // 日志输出者 输出到文件
SplitFilesLogger.cs        // 日志输出者 输出到多个文件并附上日期
MultiLogger.cs             // 日志输出者 使多个日志输出者进行日志输出
NLogger.cs                 // 日志输出者 NLogger库 在ET的服务端用到
UnityLogger.cs             // 日志输出者 输出到Unity的控制台
```

系统接口

Log

- static ILog **ILog** { get; set; }

  设置使用的日志输出者

- static bool Verbose { get; set; }

  设置是否输出 调用日期 和 调用者信息

- static void **Trace**(string msg，[object[] args])

  输出等级为1

  输出调用栈

- static void **Debug**(string msg，[object[] args])

  输出等级为2

- static void **Info**(string msg，[object[] args])

  输出等级为3

- static void **TraceInfo**(string msg)

  输出等级为3

  输出调用栈

- static void **Warning**(string msg，[object[] args])

  输出等级为4

- static void **Error**(string msg，[object[] args])

  忽略输出等级

  输出调用栈

- static void **Error**(Exception e)

  忽略输出等级

输出调用栈

- static void **Console**(string message，[object[] args])

  忽略输出等级

  在c#的终端输出

NullLogger

不输出任何日志

FileLogger

- FileLogger(string filepath = null)

  构造函数，设置日志输出的文件

  如果path为空，默认输出到Log.txt文件

SplitFilesLogger

- SplitFilesLogger(string path = null)

  构造函数，设置日志输出的文件

  如果path为空，默认输出到Logs文件夹里

MultiLogger

- MultiLogger()

  构建函数，无参数配合Apped方法使用

- MultiLogger(ILog[] loggers)

  构建函数，使传入的loggers数组都输出日志

- MultiLogger(List loggers)

  构建函数，使传入的loggers数组都输出日志

- MultiLogger Append(ILog logger)

  添加额外的日志输出者

UnityLogger

日志输出到Unity控制台

使用方法

1. 第一步，设置日志输出等级

```
Options.Instance = new Options { LogLevel = 1 };
```

2. 第二步，设置日志输出详细程度

   设置输出详细信息，包括日期，脚本名，行号，方法名

```
Log.Verbose = true;
```

3. 第二步，设置日志输出者
    1. 输出到Unity控制台

```
Log.ILog = new UnityLogger();
```

    2. 输出到文件

```
Log.ILog = new FileLogger("./Log.txt");
```

    3. 同时输出到Unity控制台和文件
        1. 方式一：

```
Log.ILog = new MultiLogger(
                new ILog[] {
                    new UnityLogger(),
                    new FileLogger("./Log.txt")
                }
            );
```

        2. 方式二：

```
Log.ILog =  new MultiLogger()
                .Append(new UnityLogger())
                .Append(new FileLogger("./Log.txt"));
```

3. 第三步，输出日志

```
Log.Trace("trace");
Log.Debug("debug");
Log.Info("info");
Log.Warning("warning");
Log.Error("error");
```

详细输出内容格式

示例:

```
[E][2022-02-19 11:05:25:658] [TestLog.cs] [59] [TestLogger]
error
  at TestLog.TestLogger (ET.ILog logger) [0x00070] in
D:\Sources\UnityProjects\ZLog\trunk\Assets\ZLogExample\TestLog.cs:59
  at TestLog.Start () [0x00049] in
D:\Sources\UnityProjects\ZLog\trunk\Assets\ZLogExample\TestLog.cs:28
```

其中

- [E] : 本日志为错误日志，除此之外还有： [T] 回溯; [D] Debug; [I] Info; [W] Warning
- [2022-02-19 11:05:25:658] : 本日志的输出日期
- [TestLog.cs] ： 本日志的被调用的脚本名
- [59] ： 本日志在脚本的59行被调用
- [TestLogger] ： 调用本日志的方法名

日志输出扩展

继承实现 ILog 接口

注意事项

1. 调用输出方法时，如果要使用Format功能，避免使用 string string int string 这个顺序的参数。

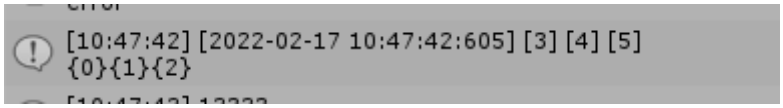
   示例:

   ```
   Log.Debug("{0}{1}{2}", "3", 4, "5");
   ```

   期望结果：输出 345

   实际结果：被当作调用者信息传入

   ![image-20220217104628962](D:\Sources\UnityProjects\ZLog\trunk\Readme\image-20220217104628962.png

应对方法: 显性使用String的Format函数

```
Log.Debug(string.Format("{0}{1}{2}", "3", 4, "5"));
```