joyo-j

# 1·登录

## 源码

### 1·反编译

#### 1·速浏定义·

using System;

namespace 接口解析类库

{

public sealed class MessageLogWriter : IDisposable

{

public MessageLogWriter() { }

public void Dispose() {throw new NotImplementedException(); }

public static MessageLogWriter Instance { get; }

public bool Started { get; }

public string CurrentFileName { get; }

}

}

##### 解析·文本显示·接口的调用

**namespace 接口解析类库**

**{**

**using System;**

**using System.Diagnostics;**

**using System.Runtime.CompilerServices;**

**public sealed class MessageLogWriter : IDisposable**

**{**

**[CompilerGenerated, DebuggerBrowsable(DebuggerBrowsableState.Never)]**

[编译生成， 可调试器可浏览（调试器可浏览状态.从不）]

**private readonly string <CurrentFileName>k\_\_BackingField;**

**[CompilerGenerated, DebuggerBrowsable(DebuggerBrowsableState.Never)]**

**private static readonly MessageLogWriter <Instance>k\_\_BackingField;**

**[CompilerGenerated, DebuggerBrowsable(DebuggerBrowsableState.Never)]**

**private readonly bool <Started>k\_\_BackingField;**

**public void Dispose()**

**{**

**throw new NotImplementedException();**

**}**

**public string CurrentFileName =>**

**this.<CurrentFileName>k\_\_BackingField;**

**public static MessageLogWriter Instance =>**

**<Instance>k\_\_BackingField;**

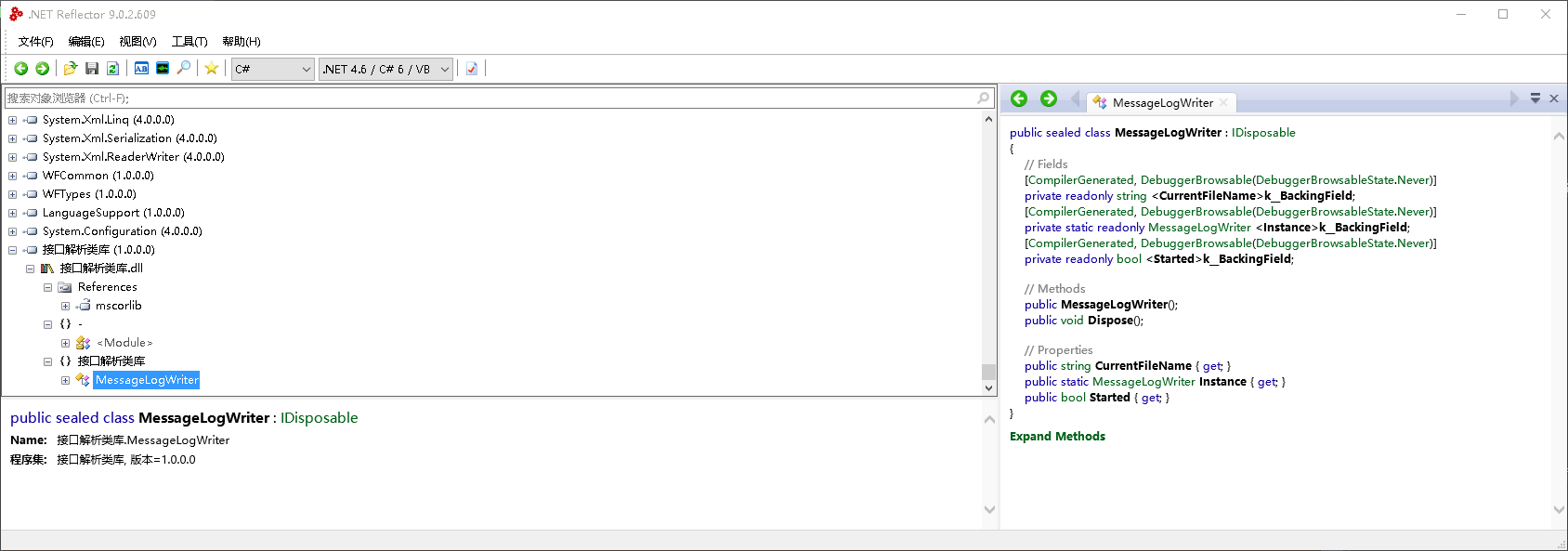
**public bool Started =>**

**this.<Started>k\_\_BackingField;**

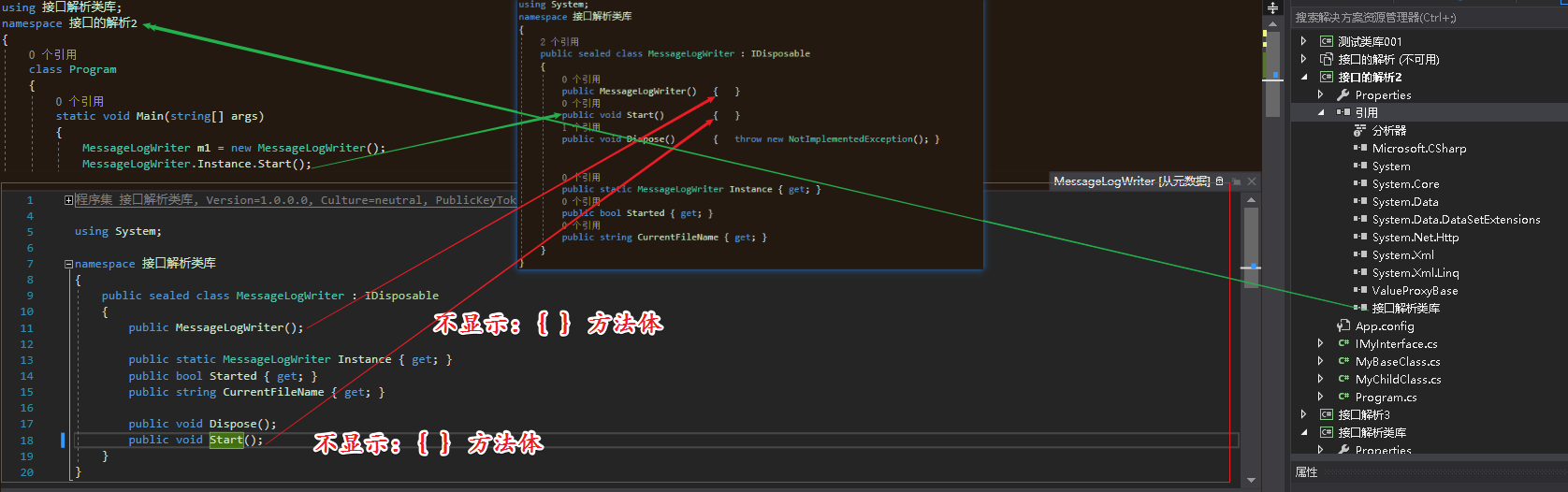
**}**

**}**

##### 解析·图片不显示·接口的调用



##### 测试调用·2个展示时候不显示



### 2·梳理

#### 1·继承父类实现接口

已经证明：

##### 子类继承·父类接口·重写接口·隐藏父类接口方法new

父类：接口1

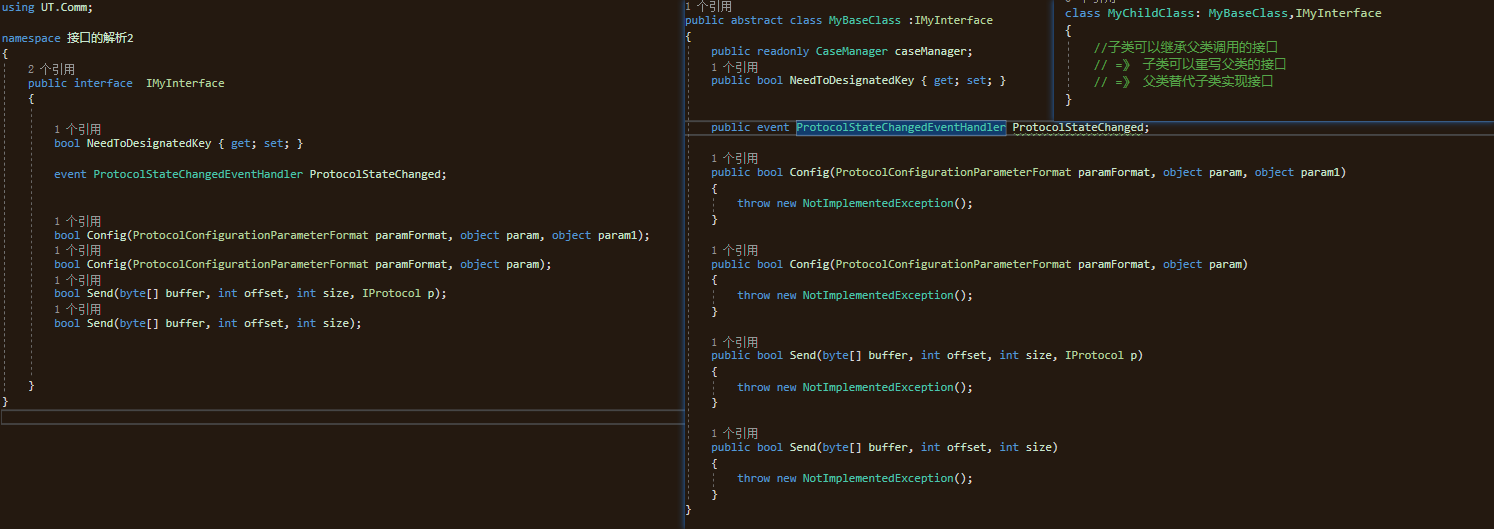
子类：父类1，

子类不需要实现父类接口了

实现完美的封装

子类·隐藏重写父类接口

代码接口的解析·2



### 3·杂项

#### 1·类实例做自己的字段 =》 该类静态实例 做属性 =》 方便在之前调用

**namespace UT.WFCommonUsed**

**{**

**//**密封的部分类·1

**public sealed class JoyoMessageLogWriter : IDisposable//**

**{**

**public JoyoMessageLogWriter();//构造函数**

**public static JoyoMessageLogWriter Instance { get; }// JoyoMessageLogWriter Instance**该类静态实例 做属性

**public bool Started { get; }**

**public string CurrentFileName { get; }**

**public void Dispose();//实现接口**

**public void InitLogFileNameSuffix(string dataPath, string fileNameSuffix);**

**public void LogMessage(string title, string message, ut.BaseTypes.PopupMessageType msgType);**

**public void LogMessage(string title, string message);**

**public void LogMessage(string fullMessage);**

**public WFCommonUsed.SystemLogFileInfo NextSystemLogFileInfo(int currentIndx);**

**public WFCommonUsed.SystemLogFileInfo PreviousSystemLogFileInfo(int currentIndx);**

**public WFCommonUsed.WFServerEventArgs ReadSystemLogFile(WFCommonUsed.SystemLogReadCommand cmd);**

**public void Start();**

**public void Stop();**

**}**

**}**

**JoyoMessageLogWriter Instance.InitLogFileNameSuffix(SystemConfig.SystemConfigFilePath,**

**string.Format(UTLanguage.Translate("卓越PC客户端日志\_{0}\_"),**

**System.Environment.MachineName));**

**//**密封的部分类·2

**public sealed class JoyoMessageLogWriter : IDisposable//**

**{**

**public static JoyoMessageLogWriter Instance**

**// JoyoMessageLogWriter Instance该类静态实例 做属性 =》**

**{**

**get**

**{**

**if (instance == null)**

**{**

**instance = new JoyoMessageLogWriter();**

**}**

**return instance;**

**}**

**}**

**}**

#### 2·private const int LogFileCount = 20;

const 无法加入断点

#### 3·密封的部分类

**public sealed partial class JoyoMessageLogWriter**

**{**

**}**

**public sealed partial class JoyoMessageLogWriter : IDisposable**

**{**

**public static JoyoMessageLogWriter Instance//自己实例做 属性**

**{**

**get**

**{**

**if (instance == null)**

**{**

**instance = new JoyoMessageLogWriter();**

**}**

**return instance;**

**}**

**}**

**}**

#### 4·index字符串截取

int tableIdx = ds.Tables.IndexOf(XmlTableName);

#### 4·winform

##### 1· this.panel1.SuspendLayout();

使用 SuspendLayout 和 ResumeLayout

许多 Windows 窗体控件（例如，ListView 和 TreeView 控件）都实现了 SuspendLayout 和 ResumeLayout 方法，

它们能够**防止控件在添加子控件时创建多个布局事件**。

如果您的控件以编程方式添加和删除子控件或者执行动态布局，则您应该调用 SuspendLayout 和 ResumeLayout 方法。通过 SuspendLayout 方法，可以在控件上执行多个操作，而不必为每个更改执行布局。例如，如果您调整控件的大小并移动控件，则每个操作都将引发单独的布局事件。这些方法按照与 BeginUpdate 和 EndUpdate 方法类似的方式操作，并且在性能和用户界面稳定性方面提供相同的好处。下面的示例以编程方式向父窗体中添加按钮：

##### 2·1((System.ComponentModel.ISupportInitialize)(this.UpDownInterval)).BeginInit();

// 摘要:

// 表示显示数值的 Windows 数字显示框（也称作 up-down 控件）。

[ClassInterface(ClassInterfaceType.AutoDispatch)]

[ComVisible(true)]

[DefaultBindingProperty("Value")]

[DefaultEvent("ValueChanged")]

[DefaultProperty("Value")]

[SRDescriptionAttribute("DescriptionNumericUpDown")]

public class NumericUpDown : UpDownBase, ISupportInitialize

###### 2·2 BeginInit()

//

// 摘要:

// 用信号通知对象初始化即将开始。

void BeginInit();

##### 3·this.panel1.Dock = System.Windows.Forms.DockStyle.Top;

// 摘要:

// 指定的位置和控件停靠的方式。

[Editor("System.Windows.Forms.Design.DockEditor, System.Design, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a", typeof(UITypeEditor))]

public enum DockStyle

##### 4` this.panel1.Dock = System.Windows.Forms.DockStyle.Top; 指定的位置和控件停靠的方式。

//

// 摘要:

// 获取或设置哪些控件边框停靠到其父控件并确定控件如何随其父级一起调整大小。

//

// 返回结果:

// System.Windows.Forms.DockStyle 值之一。 默认值为 System.Windows.Forms.DockStyle.None。

public virtual DockStyle Dock { get; set; }

5`TABLE 顺序

this.panel1.TabIndex = 3;



###### 2· this.groupBox4.TabStop = false;

获取或设置一个值，该值指示用户是否可以按 TAB 键将焦点放到

### 4·快捷

#### 1·快捷键

跳转 Ctrl + G

添加项目 Alt+F+D+E

大小写 Ctrl + Shift+U 与Ctrl + U

搜索 Ctrl + Shift +f 与Ctrl + Shift +H

#### 2·调试任务

调试技巧

初始化

Initialize

成员

static =》 readonly 静态字段

null =》 初始字段

关系

override =》 子类继承接口（父类实现）

类作为自己的实例 =》该类静态实例 做属性 =》 方便在之前调用

顺序

main =》 MainStarter.cs

构造函数