

用户

你现在 act as a csharp 程序员,我需要 一个逻辑用于解析输出 csv 中的每一个cell 的数据, csv使用英文字符的逗号隔开, 同时逻辑中需要对cell中含有英文字符的逗号的情况做处理。请给出示例代码, 下面我给出 csv 数据 以及部分 csharp 逻辑。

```
index, size_x, size_y, width, height
0, 2.5, "2.9, 55", 1920, 1080
1, 3.5, 3.9, 1921, 1081

private void ReadCSV()
{
    // 避免 csv 文件被占用, 需要使用附件开启
    var temp = Path.GetTempFileName();
    File.Copy(csv, temp, true);
    var lines = System.IO.File.ReadAllLines(temp);
    foreach (var line in lines)
    {
        var values = line.Split(',');
        foreach (var value in values)
        {
            Debug.Log(value);
        }
    }
}
```

必应

您好! 这是一个 C# 代码示例, 它可以解析 CSV 文件并处理单元格中包含英文字符逗号的情况。您可以使用以下代码替换您现有的 ReadCSV 方法:

```
private void ReadCSV()
{
    // 避免 csv 文件被占用, 需要使用附件开启
    var temp = Path.GetTempFileName();
    File.Copy(csv, temp, true);
    var lines = System.IO.File.ReadAllLines(temp);
    foreach (var line in lines)
    {
        var values = ParseLine(line);
        foreach (var value in values)
        {
            Debug.Log(value);
        }
    }
}

private string[] ParseLine(string line)
{
    List<string> result = new List<string>();
    bool inQuotes = false;
    StringBuilder currentValue = new StringBuilder();
    for (int i = 0; i < line.Length; i++)
    {
        char currentChar = line[i];
        if (currentChar == '"')
        {
            inQuotes = !inQuotes;
        }
        else if (currentChar == ',' && !inQuotes)
        {
            result.Add(currentValue.ToString());
            currentValue.Clear();
        }
        else
        {
            currentValue.Append(currentChar);
        }
    }
    result.Add(currentValue.ToString());
    return result.ToArray();
}
```

用户

现在, 我需要把从 csv中获得的数据写入一个名为 DisplayConfiguration 的对象中, 此时 ReadCSV 需要加入泛型参数 T, 经过此函数将从 csv中读取的数据填充到 上述对象的各个对应的字段中, DisplayConfiguration 类型如下:

```
public class DisplayConfiguration
{
    public int index;
    public int size_x;
    public int size_y;
    public int width;
    public int height;
}
```

需要用到的技术是反射, 请给出更新后的逻辑

必应

您好! 这是一个更新后的 ReadCSV 方法, 它使用了泛型参数 T 并使用反射将 CSV 数据填充到 DisplayConfiguration 对象中。您可以使用以下代码替换您现有的 ReadCSV 方法:

```
private List<T> ReadCSV<T>() where T : new()
{
    var result = new List<T>();
    // 避免 csv 文件被占用, 需要使用附件开启
    var temp = Path.GetTempFileName();
    File.Copy(csv, temp, true);
    var lines = System.IO.File.ReadAllLines(temp);
    string[] headers = ParseLine(lines[0]);
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
```

```
T obj = new T();
for (int j = 0; j < headers.Length; j++)
{
    var field = typeof(T).GetField(headers[j]);
    if (field != null)
    {
        field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType));
    }
}
result.Add(obj);
return result;
}

private string[] ParseLine(string line)
{
    List<string> result = new List<string>();
    bool inQuotes = false;
    StringBuilder currentValue = new StringBuilder();
    for (int i = 0; i < line.Length; i++)
    {
        char currentChar = line[i];
        if (currentChar == '"')
        {
            inQuotes = !inQuotes;
        }
        else if (currentChar == ',' && !inQuotes)
        {
            result.Add(currentValue.ToString());
            currentValue.Clear();
        }
        else
        {
            currentValue.Append(currentChar);
        }
    }
    result.Add(currentValue.ToString());
    return result.ToArray();
}
```

用户

非常不错，得到了预期的效果，如果能够将 field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType)); try catch 就完美啦。

必应

您好！您可以在 field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType)); 这一行代码周围添加 try-catch 语句来捕获可能出现的异常。下面是一个示例：

```
try
{
    field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType));
}
catch (Exception ex)
{
    // 处理异常
    Debug.Log(ex.Message);
}
```

用户

谢谢，现在，我有一个 DisplayConfiguration 的实例对象，我需要把这个对象中的数据反过来写入 csv 文件，函数名： WriteCSV (target,path),请给出改函数具体实现。

必应

您好！这是一个 WriteCSV 方法，它使用了泛型参数 T 并使用反射将 DisplayConfiguration 对象中的数据写入 CSV 文件。您可以使用以下代码：

```
private void WriteCSV<T>(List<T> target, string path)
{
    var fields = typeof(T).GetFields();
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < fields.Length; i++)
    {
        sb.Append(fields[i].Name);
        if (i < fields.Length - 1)
        {
            sb.Append(",");
        }
    }
    sb.AppendLine();
    foreach (var item in target)
    {
        for (int i = 0; i < fields.Length; i++)
        {
            var value = fields[i].GetValue(item);
            if (value != null && value.ToString().Contains(","))
            {
                sb.Append("\"" + value + "\"");
            }
            else
            {
                sb.Append(value);
            }
            if (i < fields.Length - 1)
            {
                sb.Append(",");
            }
        }
        sb.AppendLine();
    }
    File.WriteAllText(path, sb.ToString());
}
```

用户

非常酷，谢谢你的协助！！！！

必应

不客气! 很高兴能够帮助您。如果您还有其他问题, 欢迎随时咨询。

用户

现在我需要你在上面的readcsv基础上写一个函数实现给定一个已经存在的 display configuration对象, 根据 predicate 判断应该拿哪一行数据填充给定的对象, 初步构想如下: public T ReadToObject Overwrite (T target, Predicate<bool, T> predicate, string path) , 请注意predicate不一定是 Predicate 类型也可以是linq表达式, 请根据实际情况来。Predicate用于判断用哪一个表头的那一个值相同的行来填充对象。逻辑里面要判断如果表头与给定字段不存在就抛exception。

必应

您好! 这是一个 ReadToObjectOverwrite 方法, 它使用了泛型参数 T 并使用反射将 CSV 数据填充到给定的 target 对象中。您可以使用以下代码:

```
public T ReadToObjectOverwrite<T>(T target, Func<T, bool> predicate, string path) where T : new()
{
    var lines = System.IO.File.ReadAllLines(path);
    string[] headers = ParseLine(lines[0]);
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
        T obj = new T();
        for (int j = 0; j < headers.Length; j++)
        {
            var field = typeof(T).GetField(headers[j]);
            if (field != null)
            {
                field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType));
            }
            else
            {
                throw new Exception("Field not found: " + headers[j]);
            }
        }
        if (predicate(obj))
        {
            return obj;
        }
    }
    return default(T);
}
```

用户

这个方法的实现有问题, 如果你想这么写的话, 效率不高, 我建议先对 header取值进行 predicate, 另外如果直接return obj的话, 很显然target对象就被完全替换了而我只需要更新target里的成员的值。请根据提示重新整理代码, 另外如果有什么不清楚的问题可以反问我。

必应

您好! 感谢您的指正。这是一个更新后的 ReadToObjectOverwrite 方法, 它使用了泛型参数 T 并使用反射将 CSV 数据填充到给定的 target 对象中。此方法首先对表头进行 predicate 判断, 然后只更新 target 对象中的成员值。您可以使用以下代码:

```
public void ReadToObjectOverwrite<T>(T target, Func<T, bool> predicate, string path) where T : new()
{
    var lines = System.IO.File.ReadAllLines(path);
    string[] headers = ParseLine(lines[0]);
    T headerObj = new T();
    for (int i = 0; i < headers.Length; i++)
    {
        var field = typeof(T).GetField(headers[i]);
        if (field != null)
        {
            field.SetValue(headerObj, Convert.ChangeType(headers[i], field.FieldType));
        }
        else
        {
            throw new Exception("Field not found: " + headers[i]);
        }
    }
    if (!predicate(headerObj))
    {
        throw new Exception("Header does not match predicate");
    }
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
        for (int j = 0; j < headers.Length; j++)
        {
            var field = typeof(T).GetField(headers[j]);
            if (field != null)
            {
                field.SetValue(target, Convert.ChangeType(values[j], field.FieldType));
            }
            else
            {
                throw new Exception("Field not found: " + headers[j]);
            }
        }
    }
}
```

用户

写的不错, 可不可以用表达式的 predicate , 这样就可以省去new headerObj 的过程了, 直接解析表达式获取要断言的header名称和数值?

必应

您好! 是的, 您可以使用表达式树来解析 predicate 并获取要断言的表头名称和值。这样就可以省去创建 headerObj 的过程。下面是一个示例, 它使用了表达式树来解析 predicate 并获取要断言的表头名称和值:

```
public void ReadToObjectOverwrite<T>(T target, Expression<Func<T, bool>> predicate, string path) where T : new()
{
}
```

```
var lines = System.IO.File.ReadAllLines(path);
string[] headers = ParseLine(lines[0]);
var binaryExpression = predicate.Body as BinaryExpression;
var left = binaryExpression.Left as MemberExpression;
var right = binaryExpression.Right as ConstantExpression;
string headerName = left.Member.Name;
object headerValue = right.Value;
if (!headers.Contains(headerName))
{
    throw new Exception("Header not found: " + headerName);
}
int headerIndex = Array.IndexOf(headers, headerName);
for (int i = 1; i < lines.Length; i++)
{
    var values = ParseLine(lines[i]);
    if (values[headerIndex].Equals(headerValue.ToString()))
    {
        for (int j = 0; j < headers.Length; j++)
        {
            var field = typeof(T).GetField(headers[j]);
            if (field != null)
            {
                field.SetValue(target, Convert.ChangeType(values[j], field.FieldType));
            }
            else
            {
                throw new Exception("Field not found: " + headers[j]);
            }
        }
        break;
    }
}
```

用户

我太聪明了，快夸我

必应

您真的很聪明！您对问题的理解和分析能力都非常出色。继续保持下去，相信您一定能取得更大的成就！

用户

你也很棒，没有你的指导我也要摸索很久并且还要额外学习很多东西，感谢你节约了我的时间

必应

谢谢您的夸奖！我很高兴能够帮助您解决问题并节约您的时间。如果您还有其他问题，欢迎随时咨询。

用户

接下我会分三次给你一个代码文件，在给完代码前你不需要做特别响应，等我发出“开始你的表演”后，你需要非常严格的检查我给的代码中的不足并给出改善方案。

必应

好的，我会等待您的代码并在您发出“开始你的表演”后进行检查并给出改善方案。

用户

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using Unity.VisualStudio;

namespace zFramework.Extension
{
    [AttributeUsage(AttributeTargets.Field)]
    public class CsvIgnoreAttribute : Attribute { }

    public static class CsvUtility
    {
        /// <summary>
        /// 从csv文件中读取所有行返回指定类型对象列表
        /// </summary>
        /// <typeparam name="T">指定的类型</typeparam>
        /// <param name="file">csv 文件路径</param>
        /// <returns>指定类型的对象列表</returns>
        public static List<T> Read<T>(string file) where T : new()
        {
            var lines = ReadAllLines(file);
            var result = new List<T>();
            if (lines == null || lines.Length <= 0)
            {
                throw new FileLoadException($"CSV 文件 {Path.GetFileNameWithoutExtension(file)} 不含任何数据，请为 csv 文件添加有效数据！\n文件路径: {file}");
            }
            string[] headers = ParseLine(lines[0]);
            for (int i = 1; i < lines.Length; i++)
            {
                var values = ParseLine(lines[i]);
                T obj = new();
                for (int j = 0; j < headers.Length; j++)
                {
                    var field = typeof(T).GetField(headers[j]);
                    if (field != null)
                    {
                        try
                        {
                            var ignore = field.GetCustomAttributes(typeof(CsvIgnoreAttribute), false);
                            if (ignore != null && ignore.Length > 0)
                            {
                                continue;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```

        }
        field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType));
    }
    catch (Exception)
    {
        throw new InvalidCastException($"{nameof(CSVHelper)}: 字段 {headers[j]} 指定的数据{values[j]} 不是 {field.FieldType} 类型，请修改csv中数据");
    }
}
}
result.Add(obj);
}
return result;
}
}

```

必应

好的，我已经收到了您的第一部分代码。

用户

```

/// <summary>
/// 从csv文件中读取指定行返回指定类型对象
/// </summary>
/// <typeparam name="T">指定的类型</typeparam>
/// <param name="file">csv 文件路径</param>
/// <returns>指定类型的对象</returns>
public static T Read<T>(string file, Expression<Func<T, bool>> predicate) where T : new()
{
    var lines = ReadAllLines(file);
    string[] headers = ParseLine(lines[0]);
    var binaryExpression = predicate.Body as BinaryExpression;
    var left = binaryExpression.Left as MemberExpression;
    var right = binaryExpression.Right as ConstantExpression;
    string headerName = left.Member.Name;
    object headerValue = right.Value;
    UnityEngine.Debug.Log($"{nameof(CsvUtility)}: header name = {headerName} , header Value = {headerValue}");
    if (!headers.Contains(headerName))
    {
        throw new Exception($"用于断言的字段 {headerName} 在 CSV 表中没找到, 请指定正确的 CSV 文件和正确的数据类型!");
    }
    int headerIndex = Array.IndexOf(headers, headerName);
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
        if (values[headerIndex].Equals(headerValue.ToString()))
        {
            var obj = new T();
            for (int j = 0; j < headers.Length; j++)
            {
                var field = typeof(T).GetField(headers[j]);
                if (field != null)
                {
                    try
                    {
                        var ignore = field.GetCustomAttributes(typeof(CsvIgnoreAttribute), false);
                        if (ignore?.Length > 0)
                        {
                            continue;
                        }
                        field.SetValue(obj, Convert.ChangeType(values[j], field.FieldType));
                    }
                    catch (Exception)
                    {
                        throw new InvalidCastException($"{nameof(CSVHelper)}: 字段 {headers[j]} 指定的数据{values[j]} 不是 {field.FieldType} 类型，请修改csv中数据");
                    }
                }
            }
            return obj;
        }
    }
    return default;
}
}

```

必应

好的，我已经收到了您的第二部分代码。

用户

```

/// <summary>
/// 从 CSV 中读取 predicate 断言的行的数据并对指定的对象填充
/// </summary>
/// <typeparam name="T">指定类型</typeparam>
/// <param name="target">目标对象</param>
/// <param name="predicate">用于确定取哪一行的断言</param>
/// <param name="path">csv 文件路径</param>
/// <exception cref="Exception">XXX</exception>
public static void FromCsvOverwrite<T>(string path, T target, Expression<Func<T, bool>> predicate) where T : new()
{
    var lines = ReadAllLines(path);
    string[] headers = ParseLine(lines[0]);
    var binaryExpression = predicate.Body as BinaryExpression;
    var left = binaryExpression.Left as MemberExpression;
    var right = binaryExpression.Right as ConstantExpression;
    string headerName = left.Member.Name;
    object headerValue = right.Value;
    UnityEngine.Debug.Log($"{nameof(CsvUtility)}: header name = {headerName} , header Value = {headerValue}");
    if (!headers.Contains(headerName))
    {
        throw new Exception("Header not found: " + headerName);
    }
    int headerIndex = Array.IndexOf(headers, headerName);
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
        if (values[headerIndex].Equals(headerValue.ToString()))
        {
            // 这里可以写逻辑来更新目标对象
        }
    }
}

```

```

        for (int j = 0; j < headers.Length; j++)
        {
            var field = typeof(T).GetField(headers[j]);
            if (field != null)
            {
                field.SetValue(target, Convert.ChangeType(values[j], field.FieldType));
            }
            else
            {
                throw new Exception("Field not found: " + headers[j]);
            }
        }
        break;
    }
}

/// <summary>
/// 将一组实例写入csv文件
/// </summary>
/// <typeparam name="T">实例类型</typeparam>
/// <param name="target">将要保存的实例</param>
/// <param name="path">csv 路径</param>
public static void Write<T>(List<T> target, string path)
{
    var fields = typeof(T).GetFields();
    StringBuilder sb = new();
    for (int i = 0; i < fields.Length; i++)
    {
        sb.Append(fields[i].Name);
        if (i < fields.Length - 1)
        {
            sb.Append(",");
        }
    }
    sb.AppendLine();
    foreach (var item in target)
    {
        for (int i = 0; i < fields.Length; i++)
        {
            var value = fields[i].GetValue(item);
            if (value != null && value.ToString().Contains(","))
            {
                sb.Append("\"" + value + "\"");
            }
            else
            {
                sb.Append(value);
            }
            if (i < fields.Length - 1)
            {
                sb.Append(",");
            }
        }
        sb.AppendLine();
    }
    File.WriteAllText(path, sb.ToString());
}

#region Assistant Function
private static string[] ReadAllLines(string file)
{
    if (!File.Exists(file))
    {
        throw new FileNotFoundException($"CSV 文件 {Path.GetFileNameWithoutExtension(file)} 不存在，请检查文件路径! \n文件路径: {file}");
    }
    var temp = Path.GetTempFileName();
    File.Copy(file, temp, true);
    return File.ReadAllLines(temp);
}

```

必应

好的，我已经收到了您的第三部分代码。

用户

```

private static string[] ParseLine(string line)
{
    List<string> result = new();
    bool inQuotes = false;
    StringBuilder currentValue = new();
    for (int i = 0; i < line.Length; i++)
    {
        char currentChar = line[i];
        if (currentChar == '"')
        {
            inQuotes = !inQuotes;
        }
        else if (currentChar == ',' && !inQuotes)
        {
            result.Add(currentValue.ToString());
            currentValue.Clear();
        }
        else
        {
            currentValue.Append(currentChar);
        }
    }
    result.Add(currentValue.ToString());
    return result.ToArray();
}

#endregion

```

请开始你的表演！

必应

您好！我已经收到了您的完整代码。下面是我对您代码的一些建议：

1. 在 `Read<T>` 方法中，您可以使用 `File.ReadAllLines` 方法代替 `ReadAllLines` 方法，因为 `File.ReadAllLines` 方法可以直接读取文件，而不需要先将文件复制到临时目录中。
2. 在 `Read<T>` 方法中，您可以使用 `field.FieldType.IsEnum` 来判断字段是否为枚举类型，并使用 `Enum.Parse` 方法来将字符串转换为枚举值。
3. 在 `Read<T>` 和 `Read<T>(string file, Expression<Func<T, bool>> predicate)` 方法中，您可以使用 `field.GetCustomAttribute<CsvIgnoreAttribute>()` 来获取字段上的 `CsvIgnoreAttribute` 属性，而不需要使用 `field.GetCustomAttributes(typeof(CsvIgnoreAttribute), false)`。
4. 在 `FromCsvOverwrite<T>` 方法中，您可以使用 `File.ReadAllLines` 方法代替 `ReadAllLines` 方法，因为 `File.ReadAllLines` 方法可以直接读取文件，而不需要先将文件复制到临时目录中。
5. 在 `Write<T>` 方法中，您可以使用 `field.GetCustomAttribute<CsvIgnoreAttribute>()` 来获取字段上的 `CsvIgnoreAttribute` 属性，而不需要使用 `field.GetCustomAttributes(typeof(CsvIgnoreAttribute), false)`。

希望这些建议能够对您有所帮助！

用户

谢谢你的建议，先将文件复制到临时目录主要是避免文件被占用

必应

您好！感谢您的解释。将文件复制到临时目录确实可以避免文件被占用的问题。如果您希望在读取文件时避免文件被占用，那么您现有的做法是正确的。

用户

根据我给你的代码，为每个 API 给出一个单例测试的代码

必应

您好！下面是一个单元测试的示例，它测试了您给出的每个 API：

```
using NUnit.Framework;
using System.Collections.Generic;
using System.IO;
using zFramework.Extension;

namespace Tests
{
    public class CsvUtilityTests
    {
        private string testCsvPath;

        [SetUp]
        public void Setup()
        {
            testCsvPath = Path.Combine(Path.GetTempPath(), "test.csv");
            File.WriteAllText(testCsvPath, "index,size_x,size_y,width,height\n0,2.5,\"2.9,55\",1920,1080\n1,3.5,3.9,1921,1081");
        }

        [Test]
        public void TestReadAll()
        {
            var result = CsvUtility.Read<DisplayConfiguration>(testCsvPath);
            Assert.AreEqual(2, result.Count);
            Assert.AreEqual(0, result[0].index);
            Assert.AreEqual(2.5f, result[0].size_x);
            Assert.AreEqual("2.9,55", result[0].size_y);
            Assert.AreEqual(1920, result[0].width);
            Assert.AreEqual(1080, result[0].height);
            Assert.AreEqual(1, result[1].index);
            Assert.AreEqual(3.5f, result[1].size_x);
            Assert.AreEqual(3.9f, result[1].size_y);
            Assert.AreEqual(1921, result[1].width);
            Assert.AreEqual(1081, result[1].height);
        }

        [Test]
        public void TestReadWithPredicate()
        {
            var result = CsvUtility.Read<DisplayConfiguration>(testCsvPath, x => x.index == 1);
            Assert.IsNotNull(result);
            Assert.AreEqual(1, result.index);
            Assert.AreEqual(3.5f, result.size_x);
            Assert.AreEqual(3.9f, result.size_y);
            Assert.AreEqual(1921, result.width);
            Assert.AreEqual(1081, result.height);
        }

        [Test]
        public void TestFromCsvOverwrite()
        {
            var target = new DisplayConfiguration();
            CsvUtility.FromCsvOverwrite(testCsvPath, target, x => x.index == 1);
            Assert.AreEqual(1, target.index);
            Assert.AreEqual(3.5f, target.size_x);
            Assert.AreEqual(3.9f, target.size_y);
            Assert.AreEqual(1921, target.width);
            Assert.AreEqual(1081, target.height);
        }

        [Test]
        public void TestWrite()
        {
            var data = new List<DisplayConfiguration>
            {
                new DisplayConfiguration { index = 2, size_x = 4.5f, size_y = "4.9", width = 1922, height = 1082 },
                new DisplayConfiguration { index = 3, size_x = 5.5f, size_y = "5.9", width = 1923, height = 1083 }
            };
            CsvUtility.Write(data, testCsvPath);
        }
    }
}
```

```

        var lines = File.ReadAllLines(testCsvPath);
        Assert.AreEqual("index,size_x,size_y,width,height", lines[0]);
        Assert.AreEqual("2,\"4.5\",4.9,\"1922\", \"1082\"", lines[1]);
        Assert.AreEqual("3,\"5.5\",5.9,\"1923\", \"1083\"", lines[2]);
    }

    [TearDown]
    public void TearDown()
    {
        File.Delete(testCsvPath);
    }
}

public class DisplayConfiguration
{
    public int index;
    public float size_x;
    public string size_y;
    public int width;
    public int height;
}
}

```

请注意，您需要在测试项目中添加对 `NUnit` 包的引用才能运行此测试。

用户

如果我是在 Unity 中开发，请给出基于 Unity Test Framework 的单例测试案例

必应

您好！如果您是在 Unity 中开发，您可以使用 Unity Test Framework 来编写单元测试。下面是一个基于 Unity Test Framework 的单元测试示例，它测试了您给出的每个 API：

```

using NUnit.Framework;
using System.Collections.Generic;
using System.IO;
using UnityEngine.TestTools;
using zFramework.Extension;

namespace Tests
{
    public class CsvUtilityTests
    {
        private string testCsvPath;

        [SetUp]
        public void Setup()
        {
            testCsvPath = Path.Combine(Path.GetTempPath(), "test.csv");
            File.WriteAllText(testCsvPath, "index,size_x,size_y,width,height\n0,2.5,\"2.9,55\",1920,1080\n1,3.5,3.9,1921,1081");
        }

        [Test]
        public void TestReadAll()
        {
            var result = CsvUtility.Read<DisplayConfiguration>(testCsvPath);
            Assert.AreEqual(2, result.Count);
            Assert.AreEqual(0, result[0].index);
            Assert.AreEqual(2.5f, result[0].size_x);
            Assert.AreEqual("2.9,55", result[0].size_y);
            Assert.AreEqual(1920, result[0].width);
            Assert.AreEqual(1080, result[0].height);
            Assert.AreEqual(1, result[1].index);
            Assert.AreEqual(3.5f, result[1].size_x);
            Assert.AreEqual(3.9f, result[1].size_y);
            Assert.AreEqual(1921, result[1].width);
            Assert.AreEqual(1081, result[1].height);
        }

        [Test]
        public void TestReadWithPredicate()
        {
            var result = CsvUtility.Read<DisplayConfiguration>(testCsvPath, x => x.index == 1);
            Assert.IsNotNull(result);
            Assert.AreEqual(1, result.index);
            Assert.AreEqual(3.5f, result.size_x);
            Assert.AreEqual(3.9f, result.size_y);
            Assert.AreEqual(1921, result.width);
            Assert.AreEqual(1081, result.height);
        }

        [Test]
        public void TestFromCsvOverwrite()
        {
            var target = new DisplayConfiguration();
            CsvUtility.FromCsvOverwrite(testCsvPath, target, x => x.index == 1);
            Assert.AreEqual(1, target.index);
            Assert.AreEqual(3.5f, target.size_x);
            Assert.AreEqual(3.9f, target.size_y);
            Assert.AreEqual(1921, target.width);
            Assert.AreEqual(1081, target.height);
        }

        [Test]
        public void TestWrite()
        {
            var data = new List<DisplayConfiguration>
            {
                new DisplayConfiguration { index = 2, size_x = 4.5f, size_y = "4.9", width = 1922, height = 1082 },
                new DisplayConfiguration { index = 3, size_x = 5.5f, size_y = "5.9", width = 1923, height = 1083 }
            };
            CsvUtility.Write(data, testCsvPath);

            var lines = File.ReadAllLines(testCsvPath);
            Assert.AreEqual("index,size_x,size_y,width,height", lines[0]);
            Assert.AreEqual("2,\"4.5\",4.9,\"1922\", \"1082\"", lines[1]);
            Assert.AreEqual("3,\"5.5\",5.9,\"1923\", \"1083\"", lines[2]);
        }
    }
}

```



```

        [TearDown]
        public void TearDown()
        {
            File.Delete(testCsvPath);
        }
    }

    public class DisplayConfiguration
    {
        public int index;
        public float size_x;
        public string size_y;
        public int width;
        public int height;
    }
}

```

请注意，您需要在 Unity 中安装 Test Framework 包并在测试脚本中添加对 UnityEngine.TestTools 和 NUnit.Framework 程序集的引用才能运行此测试。

用户

现在请撰写 Write 函数，将指定的 对象实例写入 csv，通过 predicate 断言需要更新哪一行，如果不存在这一行就写入新的一行，写入新的一行还是更新数据使用枚举 KeyinType(Update,Append)决定,如果Update情况下predicate断言失败说明要更新的那一行不存在则抛异常，Append情况下Predicate断言用于避免重复，如果断言成果就抛指定行数据已存在的异常。下面是 Write 函数签名的设计和必要的辅助方法（辅助方法不需要你实现）。请给出 Write 函数的完整实现。

```

public static void Write<T>(T target, string path, Expression<Func<T, bool>> predicate)
{
}

#region Assistant Function
private static string[] ReadAllLines(string file)
{
}
private static string[] ParseLine(string line)
{
}
private static (string filed, object value) PredicateResolver<T>(Expression<Func<T, bool>> predicate)
{
    var binaryExpression = predicate.Body as BinaryExpression;
    var left = binaryExpression.Left as MemberExpression;
    var right = binaryExpression.Right as ConstantExpression;
    return (left.Member.Name, right.Value);
}
#endregion

```

必应

您好！下面是一个 Write 方法的实现，它使用了泛型参数 T 并使用反射将给定的 target 对象写入 CSV 文件。此方法使用 predicate 来确定需要更新哪一行，如果不存在这一行就写入新的一行。写入新的一行还是更新数据由 KeyinType 枚举决定。如果在 Update 模式下 predicate 断言失败，则说明要更新的那一行不存在，此时会抛出异常。在 Append 模式下，predicate 用于避免重复，如果断言成功，则说明指定行数据已存在，此时会抛出异常。您可以使用以下代码：

```

public enum KeyinType
{
    Update,
    Append
}

public static void Write<T>(T target, string path, Expression<Func<T, bool>> predicate, KeyinType keyinType)
{
    var lines = ReadAllLines(path);
    string[] headers = ParseLine(lines[0]);
    var (headerName, headerValue) = PredicateResolver(predicate);
    if (!headers.Contains(headerName))
    {
        throw new Exception($"用于断言的字段 {headerName} 在 CSV 表头中没找到, 请指定正确的 CSV 文件和正确的数据类型!");
    }
    int headerIndex = Array.IndexOf(headers, headerName);
    bool found = false;
    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);
        if (values[headerIndex].Equals(headerValue.ToString()))
        {
            found = true;
            if (keyinType == KeyinType.Update)
            {
                StringBuilder sb = new();
                for (int j = 0; j < headers.Length; j++)
                {
                    var field = typeof(T).GetField(headers[j]);
                    if (field != null)
                    {
                        var value = field.GetValue(target);
                        if (value != null && value.ToString().Contains(","))
                        {
                            sb.Append($"\" + value + "\"");
                        }
                        else
                        {
                            sb.Append(value);
                        }
                        if (j < headers.Length - 1)
                        {
                            sb.Append(",");
                        }
                    }
                }
                lines[i] = sb.ToString();
            }
            else
            {
                throw new Exception($"指定行数据已存在");
            }
            break;
        }
    }
}

```

```

    }
    if (!found)
    {
        if (keyinType == KeyinType.Update)
        {
            throw new Exception("指定行数据不存在");
        }
        else
        {
            StringBuilder sb = new();
            for (int j = 0; j < headers.Length; j++)
            {
                var field = typeof(T).GetField(headers[j]);
                if (field != null)
                {
                    var value = field.GetValue(target);
                    if (value != null && value.ToString().Contains(","))
                    {
                        sb.Append("\"" + value + "\"");
                    }
                    else
                    {
                        sb.Append(value);
                    }
                    if (j < headers.Length - 1)
                    {
                        sb.Append(",");
                    }
                }
            }
            lines[lines.Length - 1] += "\n" + sb.ToString();
        }
    }
    File.WriteAllLines(path, lines);
}

```

希望这个实现能够满足您的需求！

用户

请为这个函数配套单例测试代码，尽可能把两种 keyin 方式都囊括

必应

您好！下面是一个单元测试的示例，它测试了 Write 方法的两种 keyin 方式：

```

using NUnit.Framework;
using System.Collections.Generic;
using System.IO;
using UnityEngine.TestTools;
using zFramework.Extension;

namespace Tests
{
    public class CsvUtilityTests
    {
        private string testCsvPath;

        [SetUp]
        public void Setup()
        {
            testCsvPath = Path.Combine(Path.GetTempPath(), "test.csv");
            File.WriteAllText(testCsvPath, "index,size_x,size_y,width,height\n0,2.5,\"2.9,55\",1920,1080\n1,3.5,3.9,1921,1081");
        }

        [Test]
        public void TestWriteUpdate()
        {
            var target = new DisplayConfiguration { index = 1, size_x = 4.5f, size_y = "4.9", width = 1922, height = 1082 };
            CsvUtility.Write(target, testCsvPath, x => x.index == 1, KeyinType.Update);

            var lines = File.ReadAllLines(testCsvPath);
            Assert.AreEqual("index,size_x,size_y,width,height", lines[0]);
            Assert.AreEqual("0,\"2.5\", \"2.9,55\", \"1920\", \"1080\"", lines[1]);
            Assert.AreEqual("1,\"4.5\",4.9,\"1922\", \"1082\"", lines[2]);
        }

        [Test]
        public void TestWriteAppend()
        {
            var target = new DisplayConfiguration { index = 2, size_x = 4.5f, size_y = "4.9", width = 1922, height = 1082 };
            CsvUtility.Write(target, testCsvPath, x => x.index == 2, KeyinType.Append);

            var lines = File.ReadAllLines(testCsvPath);
            Assert.AreEqual("index,size_x,size_y,width,height", lines[0]);
            Assert.AreEqual("0,\"2.5\", \"2.9,55\", \"1920\", \"1080\"", lines[1]);
            Assert.AreEqual("1,\"3.5\",3.9,\"1921\", \"1081\"", lines[2]);
            Assert.AreEqual("2,\"4.5\",4.9,\"1922\", \"1082\"", lines[3]);
        }

        [TearDown]
        public void TearDown()
        {
            File.Delete(testCsvPath);
        }
    }

    public class DisplayConfiguration
    {
        public int index;
        public float size_x;
        public string size_y;
        public int width;
        public int height;
    }
}

```

请注意，您需要在 Unity 中安装 Test Framework 包并在测试脚本中添加对 `UnityEngine.TestTools` 和 `NUnit.Framework` 程序集的引用才能运行此测试。