

用户

这个 GitHub 上的脚本是我的 CSV 读取工具，现在想 加入一个别名特性，计划命名为：ColumnAttribute，请把实现整合到这个脚本中去，谢谢。
<https://raw.githubusercontent.com/Bian-Sh/CsvUtility-For-Unity/master/Assets/zFramework/Extension/CsvUtility.cs>

必应

您好，我是 Bing。很抱歉，我无法直接修改您的脚本。但是，我可以提供一些建议来帮助您实现别名特性。

您可以在您的 CsvUtility 类中添加一个名为 ColumnAttribute 的类，它可以用作属性来定义列的别名。例如：

```
public class CsvUtility
{
    // ...

    [AttributeUsage(AttributeTargets.Field | AttributeTargets.Property)]
    public class ColumnAttribute : Attribute
    {
        public string Alias { get; set; }

        public ColumnAttribute(string alias)
        {
            Alias = alias;
        }
    }

    // ...
}
```

然后，您可以在您的数据类中使用此属性来定义列的别名。例如：

```
public class MyData
{
    [CsvUtility.Column("Name")]
    public string FullName { get; set; }

    [CsvUtility.Column("Age")]
    public int AgeInYears { get; set; }
}
```

最后，您需要更新 CsvUtility 类中的代码以使用此别名。例如，在读取 CSV 文件时，您可以使用反射来获取带有 ColumnAttribute 属性的字段或属性，并使用其别名来查找相应的列。

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

用户

无需你直接修改，你可以把整合后的逻辑输出到你的回复中。要求：此特性只用在 Field（字段）。

必应

好的，我明白了。您可以在 CsvUtility 类中添加一个名为 ColumnAttribute 的类，它可以用作属性来定义列的别名。由于您只希望此特性用于字段，因此您可以将 AttributeUsage 限制为仅适用于字段。下面是一个示例：

```
public class CsvUtility
{
    // ...

    [AttributeUsage(AttributeTargets.Field)]
    public class ColumnAttribute : Attribute
    {
        public string Alias { get; set; }

        public ColumnAttribute(string alias)
        {
            Alias = alias;
        }
    }

    // ...
}
```

然后，您可以在您的数据类中使用此属性来定义列的别名。例如：

```
public class MyData
{
    [CsvUtility.Column("Name")]
    public string FullName;

    [CsvUtility.Column("Age")]
    public int AgeInYears;
}
```

最后，您需要更新 CsvUtility 类中的代码以使用此别名。例如，在读取 CSV 文件时，您可以使用反射来获取带有 ColumnAttribute 属性的字段，并使用其别名来查找相应的列。

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

用户

请根据你对脚本的读取何理解，把这个实现：“在读取 CSV 文件时，您可以使用反射来获取带有 ColumnAttribute 属性的字段，并使用其别名来查找相应的列。”

必应

根据您提供的脚本链接，我查看了您的 CsvUtility 类的代码。您可以在 CsvUtility 类中添加一个名为 ColumnAttribute 的类，它可以用作属性来定义列的别名。下面是一个示例：

```
public class CsvUtility
{
    // ...

    [AttributeUsage(AttributeTargets.Field)]
    public class ColumnAttribute : Attribute
    {
        public string Alias { get; set; }

        public ColumnAttribute(string alias)
        {
            Alias = alias;
        }
    }

    // ...
}
```

然后，您可以在您的数据类中使用此属性来定义列的别名。例如：

```
public class MyData
{
    [CsvUtility.Column("Name")]
    public string FullName;

    [CsvUtility.Column("Age")]
    public int AgeInYears;
}
```

最后，您需要更新 CsvUtility 类中的代码以使用此别名。例如，在读取 CSV 文件时，您可以使用反射来获取带有 ColumnAttribute 属性的字段，并使用其别名来查找相应的列。下面是一个示例，它演示了如何在 Read 方法中实现此功能：

```
public static List<T> Read<T>(string file) where T : new()
{
    var lines = ReadAllLines(file);
    var result = new List<T>();
    if (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]);

    // 获取带有 ColumnAttribute 属性的字段
    var fields = typeof(T).GetFields().Where(f => f.GetCustomAttribute<ColumnAttribute>() != null);

    // 创建一个字典来存储字段名称和其别名之间的映射关系
    var aliasMap = new Dictionary<string, string>();
    foreach (var field in fields)
    {
        var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
        aliasMap[columnAttr.Alias] = field.Name;
    }

    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);

        // 使用别名来查找相应的列
        for (int j = 0; j < headers.Length; j++)
        {
            if (aliasMap.ContainsKey(headers[j]))
            {
                headers[j] = aliasMap[headers[j]];
            }
        }

        result.Add(SetObjectFieldData<T>(headers, values));
    }

    return result;
}
```

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

用户

请根据你看到的脚本内容和对他的理解，把 ColumnAttribute 相关处理逻辑也写到

必应

根据您提供的脚本链接

用户

可以再优化一下：

1. ColumnAttribute 不是充要条件，如果没有此特性就直接取字段名为列名。
2. 把 CsvIgnoreAttribre 考虑进去
3. 字典可以直接存 alias + FieldInfo
4. 函数 SetObjectFieldData 内的逻辑也要配套。

必应

好的，我明白了。根据您的要求，我对 CsvUtility 类的代码进行了一些修改，以实现您所需的别名特性。下面是一个示例，它演示了如何在 CsvUtility 类中添加 ColumnAttribute 类，并在 Read 方法中使用它来实现别名特性：

```
public class CsvUtility
{
    // ...
```

```

[AttributeUsage(AttributeTargets.Field)]
public class ColumnAttribute : Attribute
{
    public string Alias { get; set; }

    public ColumnAttribute(string alias)
    {
        Alias = alias;
    }
}

public static List<T> Read<T>(string file) where T : new()
{
    var lines = ReadAllLines(file);
    var result = new List<T>();
    if (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]);

    // 获取带有 ColumnAttribute 或 CsvIgnoreAttribute 属性的字段
    var fields = typeof(T).GetFields();

    // 创建一个字典来存储字段名称和其别名之间的映射关系
    var aliasMap = new Dictionary<string, FieldInfo>();
    foreach (var field in fields)
    {
        if (field.GetCustomAttribute<CsvIgnoreAttribute>() != null)
        {
            continue;
        }

        var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
        if (columnAttr != null)
        {
            aliasMap[columnAttr.Alias] = field;
        }
        else
        {
            aliasMap[field.Name] = field;
        }
    }

    for (int i = 1; i < lines.Length; i++)
    {
        var values = ParseLine(lines[i]);

        // 使用别名来查找相应的列
        for (int j = 0; j < headers.Length; j++)
        {
            if (aliasMap.ContainsKey(headers[j]))
            {
                headers[j] = aliasMap[headers[j]].Name;
            }
        }

        result.Add(SetObjectFieldData<T>(headers, values));
    }

    return result;
}

private static T SetObjectFieldData<T>(string[] headers, string[] values) where T : new()
{
    T obj = new T();

    // 获取带有 ColumnAttribute 或 CsvIgnoreAttribute 属性的字段
    var fields = typeof(T).GetFields();

    // 创建一个字典来存储字段名称和其别名之间的映射关系
    var aliasMap = new Dictionary<string, FieldInfo>();
    foreach (var field in fields)
    {
        if (field.GetCustomAttribute<CsvIgnoreAttribute>() != null)
        {
            continue;
        }

        var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
        if (columnAttr != null)
        {
            aliasMap[columnAttr.Alias] = field;
        }
        else
        {
            aliasMap[field.Name] = field;
        }
    }

    for (int i = 0; i < headers.Length; i++)
    {
        if (!aliasMap.ContainsKey(headers[i]))
            continue;

        var fieldInfo = aliasMap[headers[i]];

        if (fieldInfo.FieldType == typeof(int))
            fieldInfo.SetValue(obj, int.Parse(values[i]));

        else if (fieldInfo.FieldType == typeof(float))
            fieldInfo.SetValue(obj, float.Parse(values[i]));

        else if (fieldInfo.FieldType == typeof(double))
            fieldInfo.SetValue(obj, double.Parse(values[i]));

        else if (fieldInfo.FieldType == typeof(string))
            fieldInfo.SetValue(obj, values[i]);

        else if (fieldInfo.FieldType == typeof(bool))
            fieldInfo.SetValue(obj, bool.Parse(values[i]));
    }
}

```

```

        else if (fieldInfo.FieldType.IsEnum)
            fieldInfo.SetValue(obj, Enum.Parse(fieldInfo.FieldType, values[i]));

        else
            throw new NotSupportedException($"不支持类型: {fieldInfo.FieldType}");
    }

    return obj;
}

// ...
}

```

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

用户

干得漂亮，Read 函数写的真的不错，请做如下改善：

1. 请为 SetObjectFieldData 函数请新增 map 形参，避免冗余的 Attribute map 操作
2. field 赋值使用原来的 try catch + field.SetValue(target, Convert.ChangeType(values[j], field.FieldType)); 即可，无需形如 if (fieldInfo.FieldType == typeof(int)) fieldInfo.SetValue(obj, int.Parse(values[i])); 的逻辑。

必应

好的，我明白了。根据您的要求，我对 CsvUtility 类的代码进行了一些修改，以实现您所需的别名特性。下面是一个示例，它演示了如何在 CsvUtility 类中添加 ColumnAttribute 类，并在 Read 方法中使用它来实现别名特性：

```

public class CsvUtility
{
    // ...

    [AttributeUsage(AttributeTargets.Field)]
    public class ColumnAttribute : Attribute
    {
        public string Alias { get; set; }

        public ColumnAttribute(string alias)
        {
            Alias = alias;
        }
    }

    public static List<T> Read<T>(string file) where T : new()
    {
        var lines = ReadAllLines(file);
        var result = new List<T>();
        if (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]);

        // 获取带有 ColumnAttribute 或 CsvIgnoreAttribute 属性的字段
        var fields = typeof(T).GetFields();

        // 创建一个字典来存储字段名称和其别名之间的映射关系
        var aliasMap = new Dictionary<string, FieldInfo>();
        foreach (var field in fields)
        {
            if (field.GetCustomAttribute<CsvIgnoreAttribute>() != null)
            {
                continue;
            }

            var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
            if (columnAttr != null)
            {
                aliasMap[columnAttr.Alias] = field;
            }
            else
            {
                aliasMap[field.Name] = field;
            }
        }

        for (int i = 1; i < lines.Length; i++)
        {
            var values = ParseLine(lines[i]);

            // 使用别名来查找相应的列
            for (int j = 0; j < headers.Length; j++)
            {
                if (aliasMap.ContainsKey(headers[j]))
                {
                    headers[j] = aliasMap[headers[j]].Name;
                }
            }

            result.Add(SetObjectFieldData<T>(headers, values, aliasMap));
        }

        return result;
    }

    private static T SetObjectFieldData<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map) where T : new()
    {
        T obj = new T();

        for (int i = 0; i < headers.Length; i++)
        {
            if (!map.ContainsKey(headers[i]))
                continue;

            var fieldInfo = map[headers[i]];

            try
            {

```

```
        fieldInfo.SetValue(obj, Convert.ChangeType(values[i], fieldInfo.FieldType));
    }
    catch (Exception e)
    {
        Debug.LogError($"CsvUtility: {e.Message}");
        throw;
    }
}

return obj;
}

// ...
}
```

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

用户

请做如下改善：

- 1. ColumnAttribute 中 Alias 属性改成 name字段。
- 2. SetObjectFieldData 签名加上 T target，这样允许外部传入实例，参考如下：

```
private static T SetObjectFieldData2<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map, T target = default) where T : new()
{
    target ??= new T();
```

必应

好的，我明白了。根据

用户

请做如下改善：

- 1. ColumnAttribute 中 Alias 属性改成 name字段。
- 2. SetObjectFieldData 签名加上 T target，这样允许外部传入实例，参考如下：

```
private static T SetObjectFieldData2<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map, T target = default) where T : new()
{
    target ??= new T();
```

- 1. 请将 catch 到的异常原因写完善，参考如下：

```
throw new InvalidCastException($" {nameof(CsvUtility)}: 字段 {headers[j]} 指定的数据 {values[j]} 不是 {field.FieldType} 类型，请修改csv中数据！");
```

必应

好的，我明白了。根据您的要求，我对 CsvUtility 类的代码进行了一些修改，以实现您所需的别名特性。下面是一个示例，它演示了如何在 CsvUtility 类中添加 ColumnAttribute 类，并在 Read 方法中使用它来实现别名特性：

```
public class CsvUtility
{
    // ...

    [AttributeUsage(AttributeTargets.Field)]
    public class ColumnAttribute : Attribute
    {
        public string Name { get; set; }

        public ColumnAttribute(string name)
        {
            Name = name;
        }
    }

    public static List<T> Read
```

```
<div class="user-title">用户</div>
```

请做如下改善：

- 1. ColumnAttribute 中 Alias 属性改成 name 字段 而不是属性，且够造函数使用构造函数的块主体语法。
- 2. SetObjectFieldData 签名加上 T target，这样允许外部传入实例，参考如下：

```
private static T SetObjectFieldData2<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map, T target = default) where T : new()
{
    target ??= new T();
```

- 3. 请将 catch 到的异常原因写完善，参考如下：

```
throw new InvalidCastException($" {nameof(CsvUtility)}: 字段 {headers[j]} 指定的数据 {values[j]} 不是 {field.FieldType} 类型，请修改csv中数据！");
```

```
<div class="bot-title">必应</div>
```

好的，我明白了。根据您的要求，我对 CsvUtility 类的代码进行了一些修改，以实现您所需的别名特性。下面是一个示例，它演示了如何在 CsvUtility 类中添加 ColumnAttribute 类，

```
```csharp
public class CsvUtility
{
 // ...

 [AttributeUsage(AttributeTargets.Field)]
 public class ColumnAttribute : Attribute
 {
 public string Name;

 public ColumnAttribute(string name)
 {
 Name = name;
```

```

 }
}

public static List<T> Read<T>(string file, T target = default) where T : new()
{
 var lines = ReadAllLines(file);
 var result = new List<T>();
 if (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]);

 // 获取带有 ColumnAttribute 或 CsvIgnoreAttribute 属性的字段
 var fields = typeof(T).GetFields();

 // 创建一个字典来存储字段名称和其别名之间的映射关系
 var aliasMap = new Dictionary<string, FieldInfo>();
 foreach (var field in fields)
 {
 if (field.GetCustomAttribute<CsvIgnoreAttribute>() != null)
 {
 continue;
 }

 var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
 if (columnAttr != null)
 {
 aliasMap[columnAttr.Name] = field;
 }
 else
 {
 aliasMap[field.Name] = field;
 }
 }

 for (int i = 1; i < lines.Length; i++)
 {
 var values = ParseLine(lines[i]);

 // 使用别名来查找相应的列
 for (int j = 0; j < headers.Length; j++)
 {
 if (aliasMap.ContainsKey(headers[j]))
 {
 headers[j] = aliasMap[headers[j]].Name;
 }
 }

 result.Add(SetObjectFieldData<T>(headers, values, aliasMap, target));
 }

 return result;
}

private static T SetObjectFieldData<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map, T target = default) where T : new()
{
 target ??= new T();

 for (int i = 0; i < headers.Length; i++)
 {
 if (!map.ContainsKey(headers[i]))
 continue;

 var fieldInfo = map[headers[i]];

 try
 {
 fieldInfo.SetValue(target, Convert.ChangeType(values[i], fieldInfo.FieldType));
 }
 catch (Exception e)
 {
 throw new InvalidCastException($"typeof(CsvUtility)): 字段 {headers[i]} 指定的数据 {values[i]} 不是 {fieldInfo.FieldType} 类型，请修改csv中数据！");
 }
 }

 return target;
}

// ...
}

```

希望这些建议能够帮助您实现别名特性。如果您有其他问题，请随时告诉我。😊

## 用户

把这一段逻辑使用 Linq 实现：

```

var fields = typeof(T).GetFields();

// 创建一个字典来存储字段名称和其别名之间的映射关系
var aliasMap = new Dictionary<string, FieldInfo>();
foreach (var field in fields)
{
 if (field.GetCustomAttribute<CsvIgnoreAttribute>() != null)
 {
 continue;
 }

 var columnAttr = field.GetCustomAttribute<ColumnAttribute>();
 if (columnAttr != null)
 {
 aliasMap[columnAttr.Name] = field;
 }
 else
 {
 aliasMap[field.Name] = field;
 }
}

```

必应

您可以使用 Linq 来简化您的代码，以便更简洁地创建别名映射。下面是一个示例，它演示了如何使用 Linq 来实现相同的功能：

```
var aliasMap = typeof(T).GetFields()
 .Where(f => f.GetCustomAttribute<CsvIgnoreAttribute>() == null)
 .ToDictionary(
 f => f.GetCustomAttribute<ColumnAttribute>()?.Name ?? f.Name,
 f => f
);
```

这段代码首先使用 `Where` 方法来筛选出没有 `CsvIgnoreAttribute` 属性的字段，然后使用 `ToDictionary` 方法来创建一个字典，其中键是字段的别名（如果存在 `ColumnAttribute` 属性）或字段名称（如果不存在 `ColumnAttribute` 属性），值是字段本身。

希望这些建议能够帮助您简化您的代码。如果您有其他问题，请随时告诉我。😊

必应

很高兴你喜欢这个答案！

用户

这段逻辑有点绕，我不理解，感觉是多余的，甚至是错误的呢（因为它试图改原来的column 名称为字段名，这样暴露了字段名）？

```
// 使用别名来查找相应的列
for (int j = 0; j < headers.Length; j++)
{
 if (aliasMap.ContainsKey(headers[j]))
 {
 headers[j] = aliasMap[headers[j]].Name;
 }
}
```

必应

您好，您提到的这段代码确实有点绕。它试图将 CSV 文件中的列名（headers）替换为数据类中定义的字段名。这样做的目的是为了在后面调用 `SetObjectFieldData` 方法时，能够根据字段名来设置相应字段的值。

但是，您提到的问题是正确的。这段代码会暴露字段名，并且可能会导致一些问题。例如，如果 CSV 文件中的列名与数据类中定义的字段名不匹配，那么这段代码可能会导致错误。

因此，我建议您删除这段代码，并直接在 `SetObjectFieldData` 方法中使用别名映射来查找相应的列。下面是一个示例，它演示了如何在 `SetObjectFieldData` 方法中使用别名映射来实现此功能：

```
private static T SetObjectFieldData<T>(string[] headers, string[] values, Dictionary<string, FieldInfo> map, T target = default) where T : new()
{
 target ??= new T();

 for (int i = 0; i < headers.Length; i++)
 {
 if (!map.ContainsKey(headers[i]))
 continue;

 var fieldInfo = map[headers[i]];

 try
 {
 fieldInfo.SetValue(target, Convert.ChangeType(values[i], fieldInfo.FieldType));
 }
 catch (Exception e)
 {
 throw new InvalidCastException($"~{nameof(CsvUtility)}: 字段 {headers[i]} 指定的数据 {values[i]} 不是 {fieldInfo.FieldType} 类型，请修改csv中数据！");
 }
 }

 return target;
}
```

希望这些建议能够帮助您简化您的代码。如果您有其他问题，请随时告诉我。😊

用户

讲解一下新语法：索引运算符 `^`，形如：`lines[^1]`，还有其他形式吗？

必应

在 C# 8.0 中，引入了索引运算符 `^` 和范围运算符 `..`。这些运算符允许您更简洁地访问序列中的单个元素或范围。

索引运算符 `^` 用于指定一个索引相对于序列末尾的位置。例如，表达式 `lines[^1]` 表示访问序列 `lines` 中的最后一个元素。对于任意数字 `n`，索引 `^n` 等同于 `sequence.Length - n`。

范围运算符 `..` 用于指定一个范围的起始和结束位置。范围的起始位置是包含在内的，但结束位置是不包含在内的。这意味着起始位置包含在范围内，但结束位置不包含在范围内。例如，表达式 `lines[0..0]` 表示整个序列，与表达式 `lines[0..sequence.Length]` 相同。

您还可以使用开放式范围来指定仅有起始或结束位置的范围。例如，表达式 `lines[..]` 表示整个序列，表达式 `lines[..4]` 表示序列的前四个元素，表达式 `lines[6..]` 表示序列中从第六个元素开始到末尾的所有元素。

希望这些信息能够帮助您理解索引运算符和范围运算符。如果您有其他问题，请随时告诉我。😊

用户

`lines[^1^]` 再详细说一下，为啥后面还有个 `^`

必应

在我的回答中，`lines` 中的第二个 `^` 符号是用来表示脚注的。它与 C# 语言中的索引运算符 `^` 没有关系。

在我的回答中，我使用 [^n] 来表示脚注，其中 n 是一个数字。这些脚注用于引用我在回答中提到的事实的来源。例如，当我说“韩国的 KSTAR 设施（韩国聚变能研究所）的科学家们首次成功地维持了一个温度超过 1 亿摄氏度、持续时间为 30 秒的核聚变反应”，这里的 `` 表示这个事实来自于我之前搜索到的第 5 条结果。

希望这些信息能够帮助您理解我的回答中使用的脚注符号。如果您有其他问题，请随时告诉我。😊