Loxodon Framework TextUGUI

(English)

开发者 Clark

要求Unity 2021.3 或者更高版本

这是一个基于C#官方库修改的文本格式化插件，它通过扩展StringBuilder的AppendFormat函数，旨在避免在字符串拼接或数字转为字符串时产生垃圾回收（GC）。这样可以提高性能，特别是在对性能要求较高的场景下。

此外，插件还对Unity的UGUI进行了扩展，引入了两个新的文本控件：TemplateText和FormattableText。这两个控件支持MVVM的数据绑定特性，可以将ViewModel或值类型的对象与控件进行绑定，同时避免了值类型对象的装箱和拆箱，以最大程度地优化垃圾回收(GC)。

值得注意的是，如果使用Loxodon.Framework.TextMeshPro中的TemplateTextMeshPro或者FormattableTextMeshProUGUI控件，可以进一步减少垃圾回收（GC），完全0GC的更新游戏视图。

安装

使用 **OpenUPM** 安装(推荐)

OpenUPM 是一个开源的UPM包仓库，它支持发布第三方的UPM包，它能够自动管理包的依赖关系，推荐使用它安装本框架.

通过openupm命令安装包,要求nodejs and openupm-cli客户端的支持，如果没有安装请先安装nodejs和open-cli。

```
# 使用npm命令安装openupm-cli，如果已经安装请忽略.
npm install -g openupm-cli

#切换当前目录到项目的根目录
cd F:/workspace/New Unity Project

#安装 loxodon-framework-textugui
openupm add com.vovgou.loxodon-framework-textugui
```

修改**Packages/manifest.json**文件安装

通过修改manifest.json文件安装，不需要安装nodejs和openupm-cli客户端。在Unity项目根目录下找到Packages/manifest.json文件，在文件的scopedRegistries（没有可以自己添加）节点下添加第三方仓库package.openupm.com的配置，同时在dependencies节点下添加com.vovgou.loxodon-framework-textugui的配置，保存后切换到Unity窗口即可完成安装。

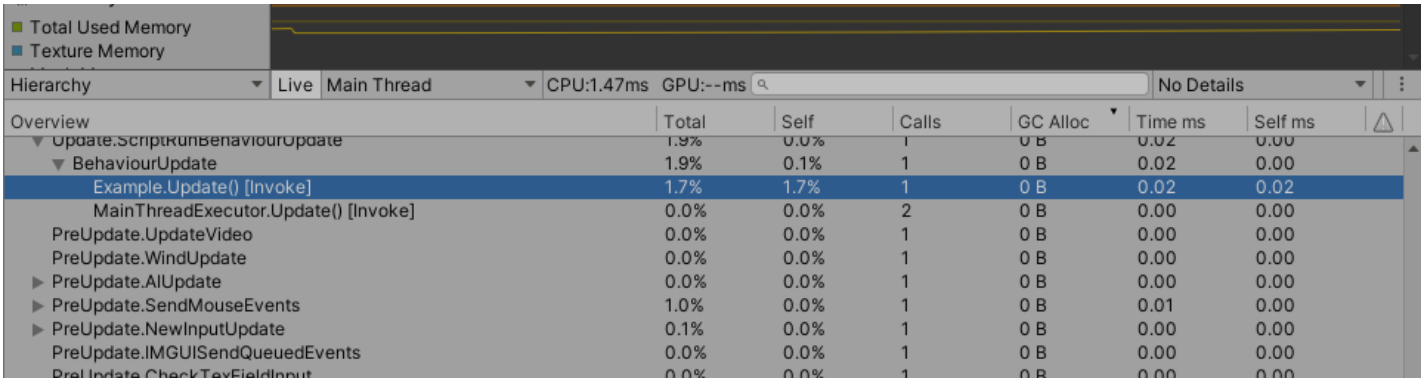
```
{
  "dependencies": {
    ...
    "com.unity.modules.xr": "1.0.0",
    "com.vovgou.loxodon-framework-textugui": "2.6.5"
  },
  "scopedRegistries": [
    {
      "name": "package.openupm.com",
      "url": "https://package.openupm.com",
      "scopes": [
        "com.vovgou",
        "com.openupm"
      ]
    }
  ]
}
```

快速开始

格式化字符串

本插件扩展了StringBuilder的AppendFormat<>()函数。支持多个不同类型的泛型参数或者泛型数组参数，当这些参数为数字类型、DateTime、TimeSpan类型时，使用它们拼接字符串时不需要将值类型装箱或者拆箱，数字类型转为String类型时不会产生垃圾回收（GC）。使用方式见下面的示例。

```
using System;
using System.Text;
using UnityEngine;
using Loxodon.Framework.TextFormatting;//必须先引入这个包名
public class Example : MonoBehaviour
{
    StringBuilder builder = new StringBuilder();
    void Update()
    {
        builder.Clear();
        builder.AppendFormat<DateTime,int>("Now:{0:yyyy-MM-dd HH:mm:ss} Frame:{0:D6}", DateTime.Now,Time.frameCount);
        builder.AppendFormat<float>("{0:f2}", Time.realtimeSinceStartup);
    }

}
```



支持格式化的文本控件(FormattableText)

此控件是UnityEngine.UI.Text的扩展，支持字符串格式化功能，支持数据绑定，FormattableText控件的AsParameters<>()函数可以转为一个泛型参数集，支持1-4个不同参数，也可以支持一个泛型数组，通过泛型参数集和ViewModel进行绑定。使用这个插件，字符串和数组拼接是无GC的，但是因为Text的text属性必须是一个字符串，并且只支持字符串赋值，所以在StringBuilder.ToString()时是有GC分配的。（建议安装Loxodon.Framework.TextMeshPro插件，使用FormattableTextMeshProUGUI替代FormattableText，那是完全0GC的）

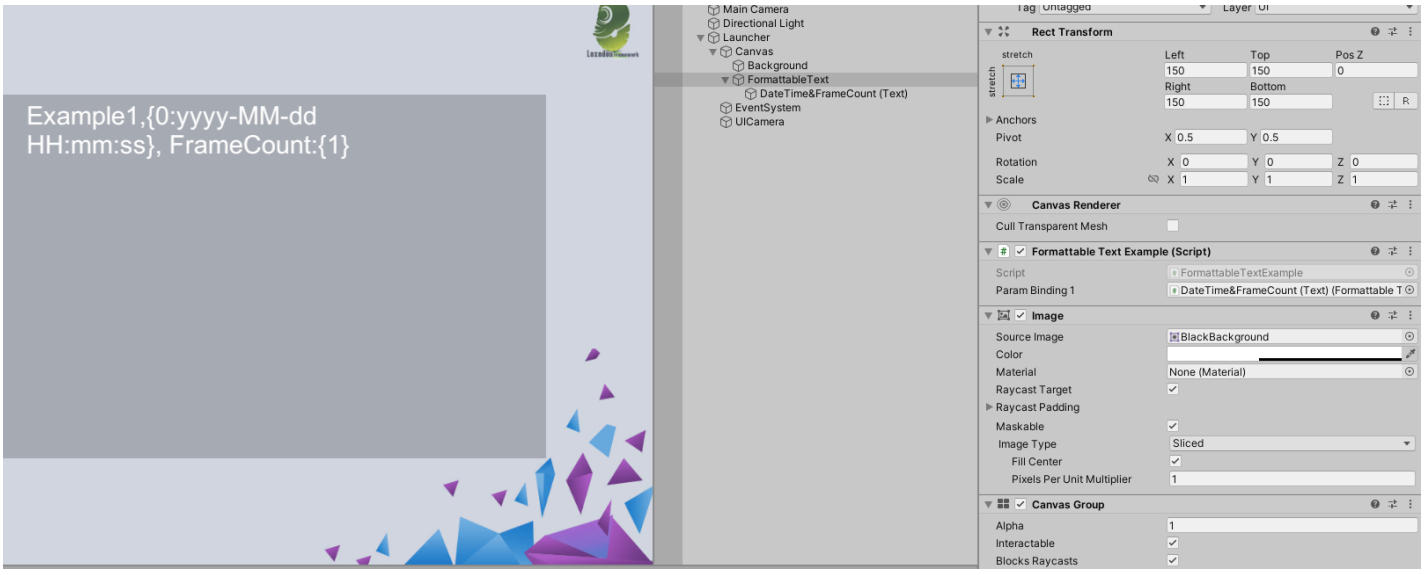
使用方式一：使用FormattableText.AsParameters<DateTime, int>()方法获得参数集GenericParameters<DateTime,int>，然后与视图模型绑定。

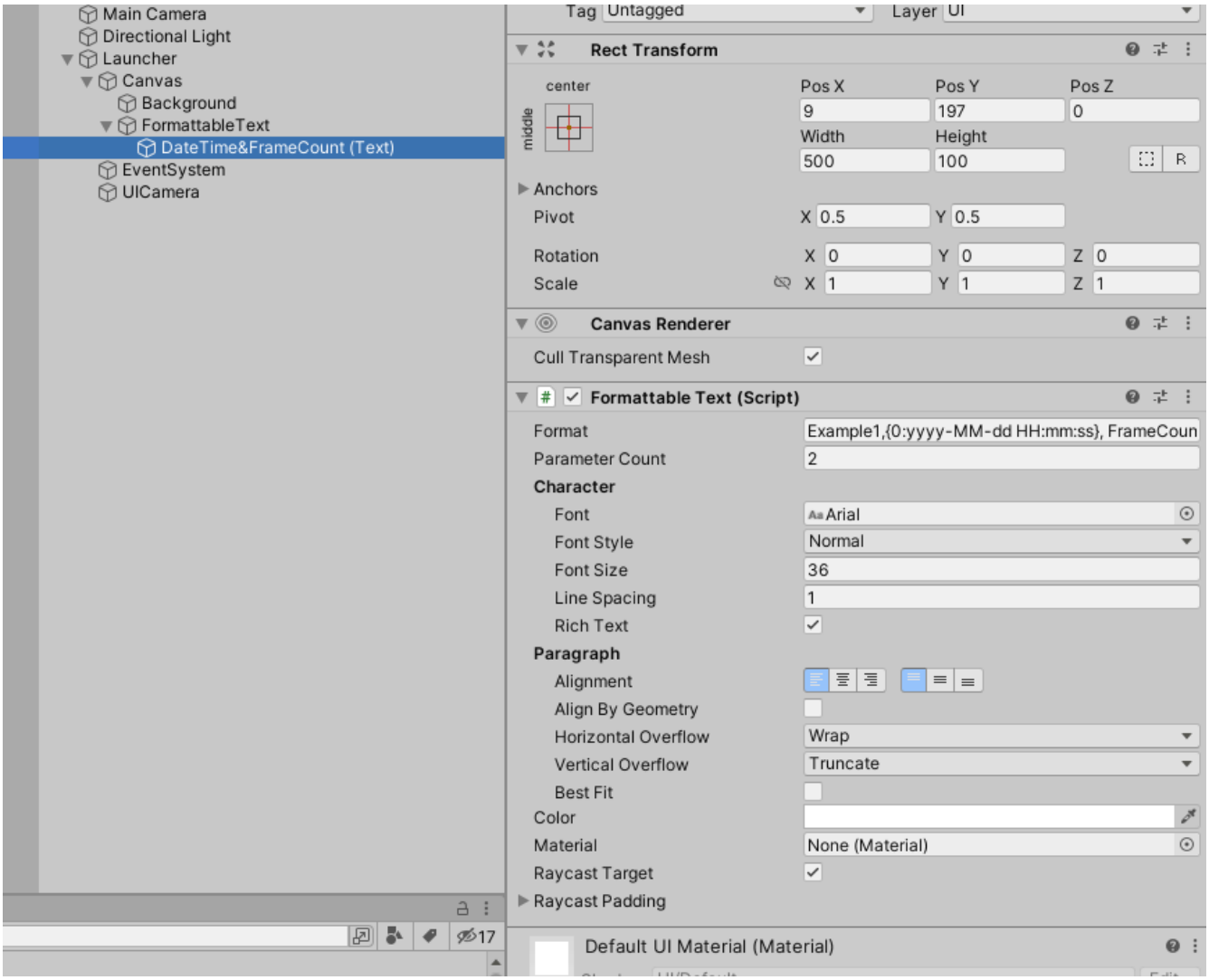
```
public class FormattableTextExample : MonoBehaviour
{
    public FormattableText paramBinding1;//参数绑定示例1，支持1-4个不同参数

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<FormattableTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<FormattableTextExample, ExampleVie

        //使用AsParameters<P1,P2,...>() 函数创建一个参数集合，然后绑定，支持1-4个参数，没有值对象的装箱拆箱，没有字符串拼接，降低GC，使用TMP文本
        //format:格式与string.Format()的格式化参数相同如：DateTime:Example1,{0:yyyy-MM-dd HH:mm:ss}, FrameCount:{1}
```

```
        bindingSet.Bind(paramBinding1.AsParameters<DateTime, int>()).For(v => v.Parameter1).To(vm => vm.Time);
        bindingSet.Bind(paramBinding1.AsParameters<DateTime, int>()).For(v => v.Parameter2).To(vm => vm.FrameCount);

        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Time = DateTime.Now;
        this.viewModel.FrameCount = 1;
        this.SetDataContext(this.viewModel);
    }
}
```

使用方式二：在脚本FormattableTextExample中定义一个类型为GenericParameters<DateTime,int>的参数集变量，在UnityEditor中将FormattableText拖放到下图脚本的属性paramBinding1上。然后与视图模型绑定。

```
public class FormattableTextExample : MonoBehaviour
{
    public GenericParameters<DateTime,int> paramBinding1;//参数绑定示例1，支持1-4个不同参数

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<FormattableTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<FormattableTextExample, ExampleVie

        //使用AsParameters<P1,P2,...>() 函数创建一个参数集合，然后绑定，支持1-4个参数，没有值对象的装箱拆箱，没有字符串拼接，降低GC，使用TMP文本
        //format:格式与string.Format()的格式化参数相同如：DateTime:Example1,{0:yyyy-MM-dd HH:mm:ss}, FrameCount:{1}
        bindingSet.Bind(paramBinding1).For(v => v.Parameter1).To(vm => vm.Time);
        bindingSet.Bind(paramBinding1).For(v => v.Parameter2).To(vm => vm.FrameCount);

        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Time = DateTime.Now;
        this.viewModel.FrameCount = 1;
        this.SetDataContext(this.viewModel);
    }
}
```





文本模版控件（TemplateText)

这个控件比格式化文本控件更强大，更好用，支持将一个ViewModel对象或者子对象绑定到TemplateText.Data属性，模版控件内置了路径解析和数据绑定功能，能自动通过文本模板来绑定Data对象中的属性。

模版格式：Template,Frame:{FrameCount:D6},Health:{Hero.Health:D4} AttackDamage:{Hero.AttackDamage} Armor:{Hero.Armor}

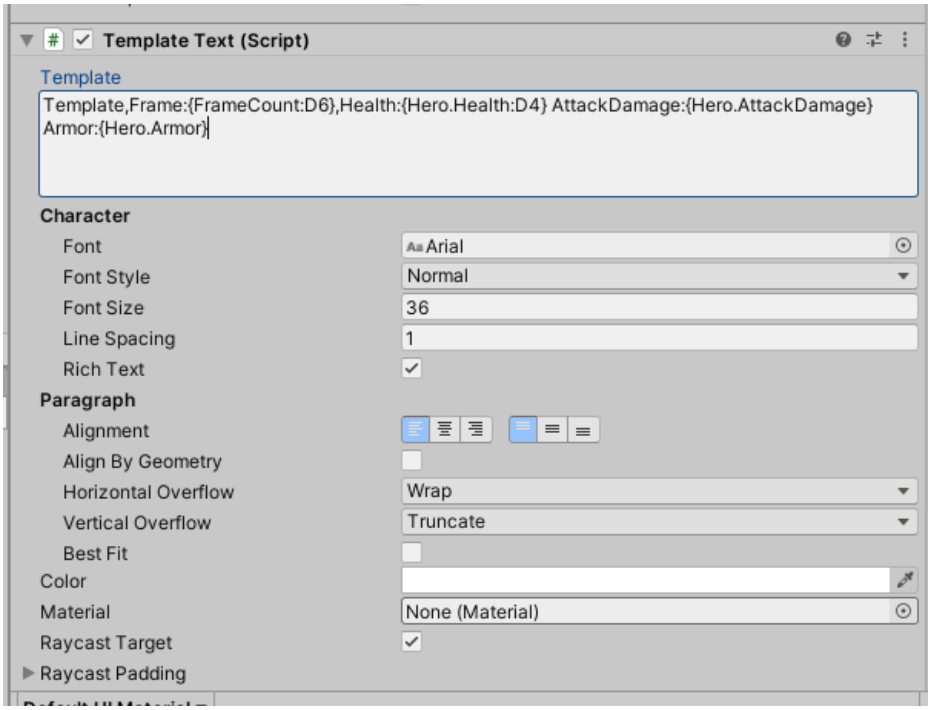
其中FrameCount、Hero是绑定到Data的对象的属性。Health、AttackDamage、Armor是Hero对象的属性。FrameCount后面的D6是帧数这个数字类型的格式化参数。

同样的只有在StringBuilder.ToString()时会产生垃圾回收(GC)。（建议安装Loxodon.Framework.TextMeshPro，使用TemplateTextMeshProUGUI替代TemplateText，那是完全0GC的）

```
public class TemplateTextExample : MonoBehaviour
{
    public TemplateText template;//模版绑定

    private ExampleViewModel viewModel;

    private void Start()
    {
        ApplicationContext context = Context.GetApplicationContext();
        IServiceContainer container = context.GetContainer();
        BindingServiceBundle bundle = new BindingServiceBundle(context.GetContainer());
        bundle.Start();

        BindingSet<TemplateTextExample, ExampleViewModel> bindingSet = this.CreateBindingSet<TemplateTextExample, ExampleViewModel
```

```
        //使用文本模版（TemplateText）绑定，直接将一个对象绑定到模板的Data属性上即可。
        //文本模版格式与string.Format类似，仅需要将{0},{1}中的数字，替换为对象属性名即可
        bindingSet.Bind(template).For(v => v.Template).To(vm => vm.Template);//模版可以绑定，也可以在编辑器上配置
        bindingSet.Bind(template).For(v => v.Data).To(vm => vm);
        bindingSet.Build();

        this.viewModel = new ExampleViewModel();
        this.viewModel.Template = "Template,Frame:{FrameCount:D6},Health:{Hero.Health:D4} AttackDamage:{Hero.AttackDamage} Armor:{
        this.viewModel.Time = DateTime.Now;
        this.viewModel.TimeSpan = TimeSpan.FromSeconds(0);
        this.viewModel.Hero = new Hero();
        this.SetDataContext(this.viewModel);
    }
}

//下面的类是ViewModel对象
public class ExampleViewModel : ObservableObject
{
    private DateTime time;
    private TimeSpan timeSpan;
    private string template;
    private int frameCount;
    private Hero hero;
    public DateTime Time
    {
        get { return this.time; }
        set { this.Set(ref time, value); }
    }

    public TimeSpan TimeSpan
    {
        get { return this.timeSpan; }
        set { this.Set(ref timeSpan, value); }
    }

    public int FrameCount
    {
        get { return this.frameCount; }
        set { this.Set(ref frameCount, value); }
    }

    public string Template
    {
        get { return this.template; }
        set { this.Set(ref template, value); }
    }

    public Hero Hero
    {
        get { return this.hero; }
        set { this.Set(ref hero, value); }
    }
}

public class Hero : ObservableObject
{
    private float attackSpeed = 95.5f;
    private float moveSpeed = 2.4f;
    private int health = 100;
    private int attackDamage = 20;
    private int armor = 30;

    public float AttackSpeed
    {
        get { return this.attackSpeed; }
        set { this.Set(ref attackSpeed, value); }
```

```
    }

    public float MoveSpeed
    {
        get { return this.moveSpeed; }
        set { this.Set(ref moveSpeed, value); }
    }

    public int Health
    {
        get { return this.health; }
        set { this.Set(ref health, value); }
    }

    public int AttackDamage
    {
        get { return this.attackDamage; }
        set { this.Set(ref attackDamage, value); }
    }

    public int Armor
    {
        get { return this.armor; }
        set { this.Set(ref armor, value); }
    }

}
```



联系方式

邮箱: yangpc.china@gmail.com
网站: https://vovgou.github.io/loxodon-framework/
QQ群: 622321589