

Trinity

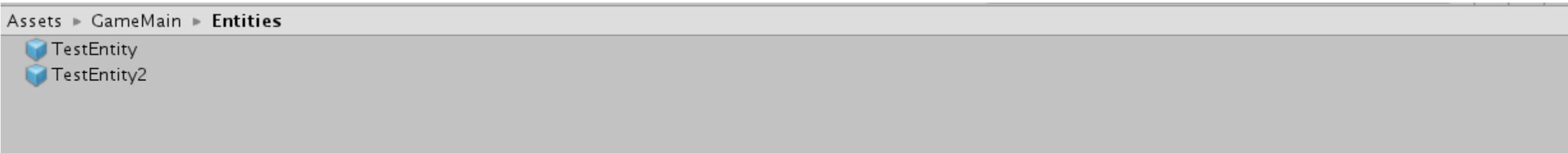
基于Unity的纯C#（客户端+服务端+热更新）游戏开发整合方案
客户端：<https://github.com/EllanJiang/UnityGameFramework>
服务端：<https://github.com/egametang/ET>
热更新：<https://github.com/Ourpalm/ILRuntime>

建议对以上三个项目有一定的了解后再使用本项目

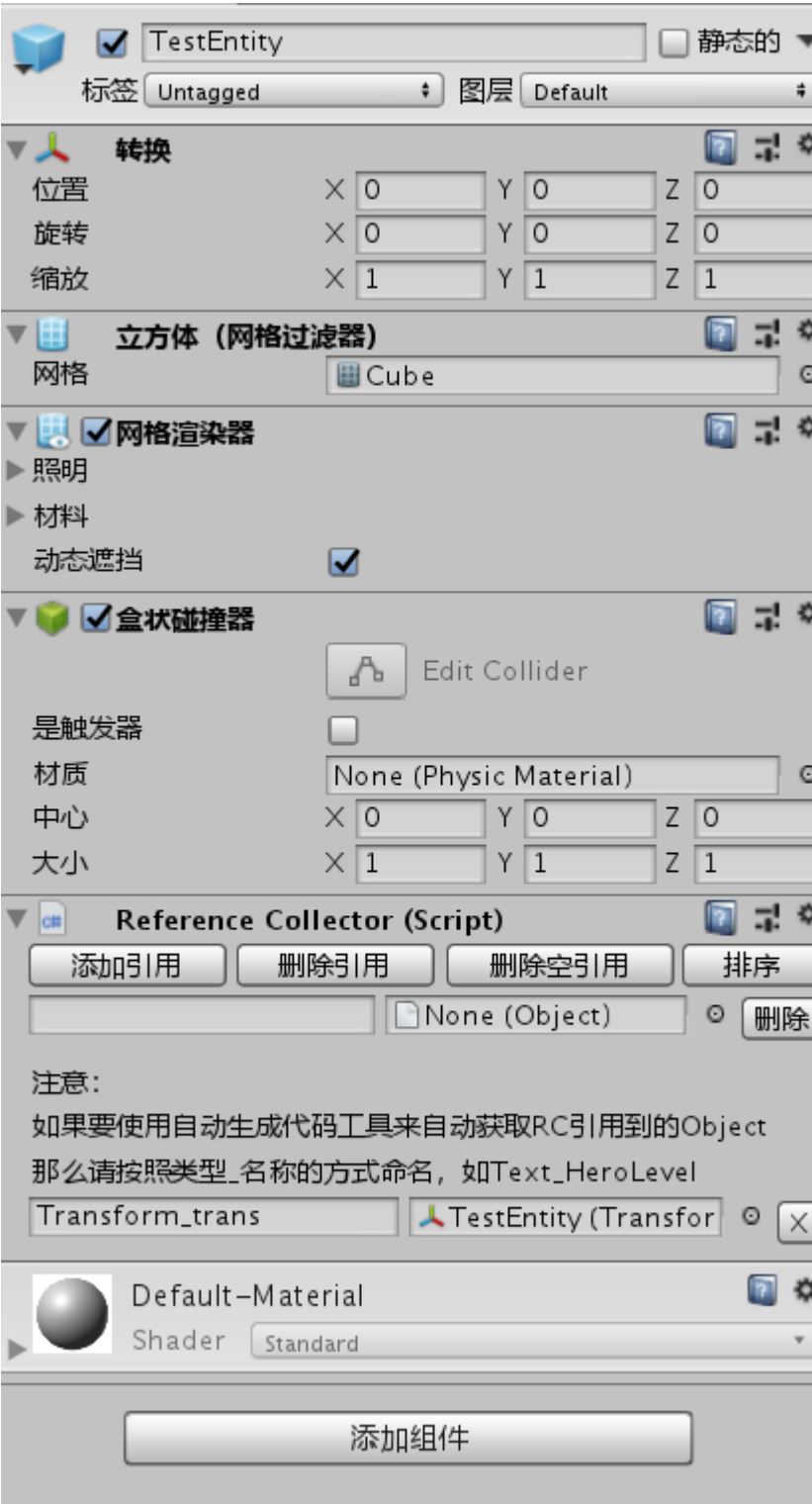
目前已完成客户端+热更新的整合，并提供了代码自动生成工具

代码自动生成工具的使用

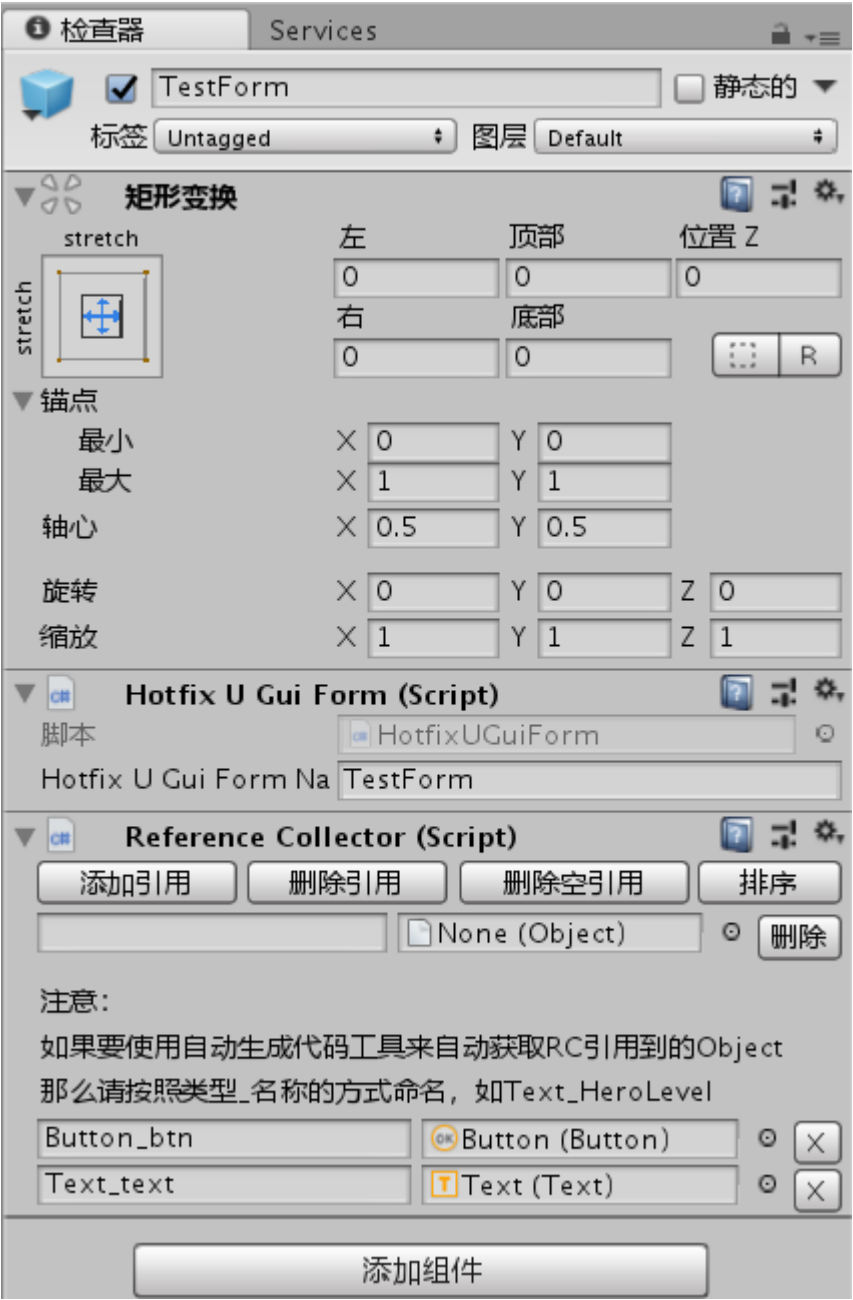
首先新建实体或界面的GameObject，并将其保存到Assets/GameMain/Entities目录下，或Assets/GameMain/UIForms目录下



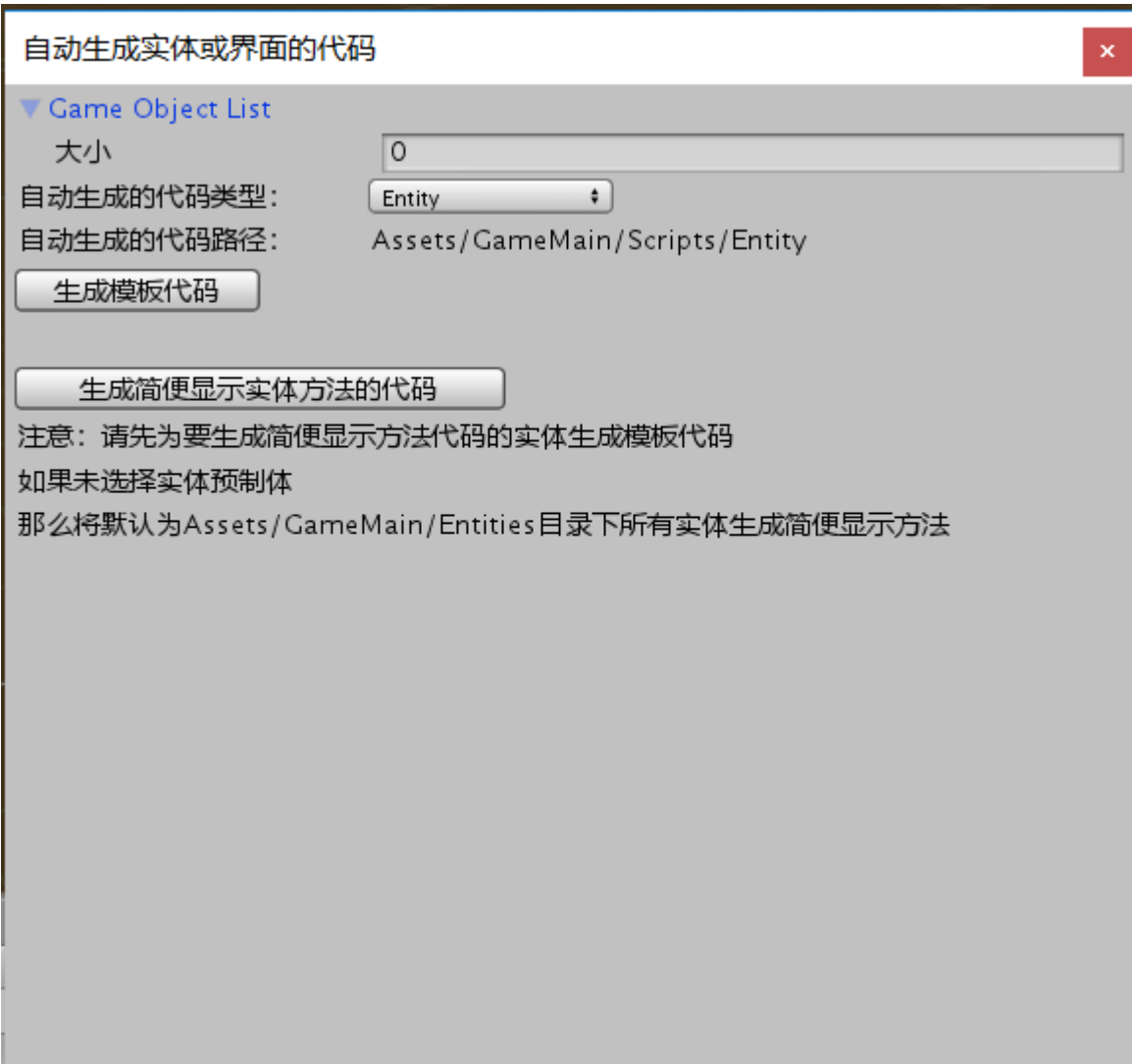
对于**并非是通过热更新来增加的实体或界面**，可以为其添加ReferenceCollector脚本来收集引用，以便于在代码中获取引用，如果想要代码自动生成工具生成获取RC上引用的Object的代码（如果要获取其他物体上的组件，可以通过开启两个Inspector面板来将其他物体的组件拖到RC上），那么需要按照指定的格式来命名



对于**并非是通过热更新来增加的界面**，需要为其添加HotfixUGuiForm脚本，并在被序列化到面板上的HotfixUGuiFormName字段上输入对应的**热更新层界面逻辑类名称**



点击编辑器菜单栏的Trinity—自动生成实体或界面的代码，会弹出对应窗口



将实体或界面的预制体拖入到GameObjectList中，选择要生成的代码类型（总共有4种：实体，热更新实体，界面，热更新界面），然后点击生成模板代码

对于实体将会生成**实体数据与实体逻辑**代码

```
using UnityEngine;

//自动生成于: 2018/10/28 12:28:22
namespace Trinity
{
    public class TestEntityData : EntityData
    {
        public TestEntityData()
        {
        }

        public TestEntityData Fill(int typeId)
        {
            Fill(GameEntry.Entity.GenerateSerialId(), typeId);
            return this;
        }

        public override void Clear()
        {
            base.Clear();
        }
    }
}
```

```
using UnityEngine;
using GameFramework;

//自动生成于: 2018/10/28 12:28:22
namespace Trinity
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class TestEntity : Entity
    {
        private TestEntityData m_TestEntityData;
        private Transform m_Transform_trans;
        7 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void OnInit(object userData)
        {
            base.OnInit(userData);

            ReferenceCollector rc = GetComponent<ReferenceCollector>();

            m_Transform_trans = rc.Get<Transform>("m_Transform_trans");
        }

        7 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void OnShow(object userData)
        {
            base.OnShow(userData);
            m_TestEntityData = (TestEntityData)userData;
        }

        7 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void OnHide(object userData)
        {
            base.OnHide(userData);
            ReferencePool.Release(m_TestEntityData);
        }
    }
}
```

```
using UnityEngine;

//自动生成于: 2018/10/28 12:33:17
namespace Trinity.Hotfix
{
    4 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class TestEntity2Data : HotfixEntityData
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TestEntity2Data()
        {
        }

        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public TestEntity2Data Fill(int typeId)
        {
            Fill(GameEntry.Entity.GenerateSerialId(), typeId);
            return this;
        }

        61 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public override void Clear()
        {
            base.Clear();
        }
    }
}

using UnityEngine;

//自动生成于: 2018/10/28 12:33:17
namespace Trinity.Hotfix
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class TestEntity2 : HotfixEntity
    {
        private TestEntity2Data m_TestEntity2Data;
        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public override void OnShow(object userData)
        {
            base.OnShow(userData);
            m_TestEntity2Data = (TestEntity2Data)userData;
        }

        2 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public override void OnHide(object userData)
        {
            base.OnHide(userData);
            ReferencePool.Release(m_TestEntity2Data);
        }
    }
}
```

对于界面将会生成**界面逻辑**代码

```
using UnityEngine;
using UnityEngine.UI;

//自动生成于: 2018/10/28 12:35:00
namespace Trinity
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class TestForm : UGuiForm
    {
        private Text m_Text_text;
        private Button m_Button_btn;
        8 个引用 | 0 项更改 | 0 名作者, 0 项更改
        protected override void OnInit(object userData)
        {
            base.OnInit(userData);

            ReferenceCollector rc = GetComponent<ReferenceCollector>();

            m_Text_text = rc.Get<Text>("m_Text_text");
            m_Button_btn = rc.Get<Button>("m_Button_btn");
        }
    }
}

using UnityEngine;
using UnityEngine.UI;

//自动生成于: 2018/10/28 12:35:12
namespace Trinity.Hotfix
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public class TestForm2 : HotfixUGuiForm
    {
    }
}
```

当已经为实体生成过模板代码后，可以点击生成实体简便显示方法的代码，为选择的实体生成简便显示方法（如果未选择任何实体则默认为Assets/GameMain/Entities目录下所有实体生成简便显示方法），此时仍然需要选择生成代码类型，但只有选择Entity或HotfixEntity才是有效的

```
using UnityGameFramework.Runtime;

//自动生成于: 2018/10/28 12:45:46
namespace Trinity
{
    public static class ShowEntityExtension
    {
        public static void ShowTestEntity(this EntityComponent entityComponent, TestEntityData data)
        {
            entityComponent.ShowEntity(typeof(TestEntity), "TestEntity", 0, data);
        }
    }
}

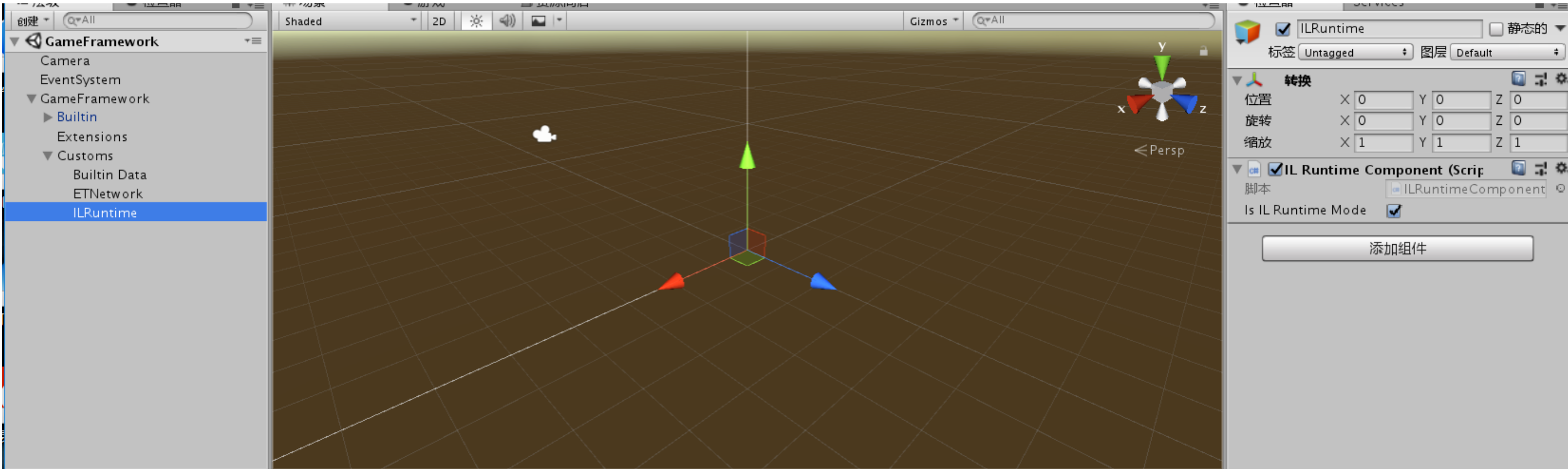
using UnityGameFramework.Runtime;

//自动生成于: 2018/10/28 12:46:12
namespace Trinity.Hotfix
{
    0 个引用 | 0 项更改 | 0 名作者, 0 项更改
    public static class ShowEntityExtension
    {
        0 个引用 | 0 项更改 | 0 名作者, 0 项更改
        public static void ShowTestEntity2(this EntityComponent entityComponent, TestEntity2Data data)
        {
            Trinity.HotfixEntityData tData = ReferencePool.Acquire<Trinity.HotfixEntityData>();
            tData.Fill(data.Id, data.TypeId, "TestEntity2", data);
            tData.Position = data.Position;
            tData.Rotation = data.Rotation;

            entityComponent.ShowHotfixEntity("TestEntity2", 0, tData);
        }
    }
}
```

热更新代码

找到ILRuntime组件，并将IsILRuntimeMode打勾



对于显示通过热更新增加的新实体，是与平常一样的显示方式，但是对于显示通过热更新增加的新界面，则需要做一些特殊处理

```
namespace Trinity.Hotfix
{
    2 个引用 | CatImmortal, 11 小时前 | 1 名作者, 1 项更改
    public class ProcedureHotfixTest : ProcedureBase
    {
        11 个引用 | CatImmortal, 11 小时前 | 1 名作者, 1 项更改
        protected internal override void OnEnter(IFsm procedureOwner)
        {
            base.OnEnter(procedureOwner);

            Log.Info("进入了热更新测试流程");



            //监听UI打开成功事件
            GameEntry.Event.Subscribe(OpenUIFormSuccessEventArgs.EventId, OnOepnUI);

            //打开UI
            GameEntry.UI.OpenUIForm(2, this);
        }

        1 个引用 | 0 项更改 | 0 名作者, 0 项更改
        private void OnOepnUI(object sender, GameEventArgs e)
        {
            OpenUIFormSuccessEventArgs ne = (OpenUIFormSuccessEventArgs)e;
            if (ne.UserData != this)
            {
                return;
            }

            //为打开的UI动态添加脚本，并执行OnHotfixInit，将对应的热更新层界面逻辑类名传递进去
            ne.UIForm.gameObject.GetOrAddComponent<Trinity.HotfixUGuiForm>().OnHotfixInit("TestForm2");
        }
    }
}
```


当编写完热更新工程里的代码后，需要先点击生成解决方案，然后点击菜单栏中的Trinity—构建热更新DLL，才能通过ILRuntime加载到热更新代码

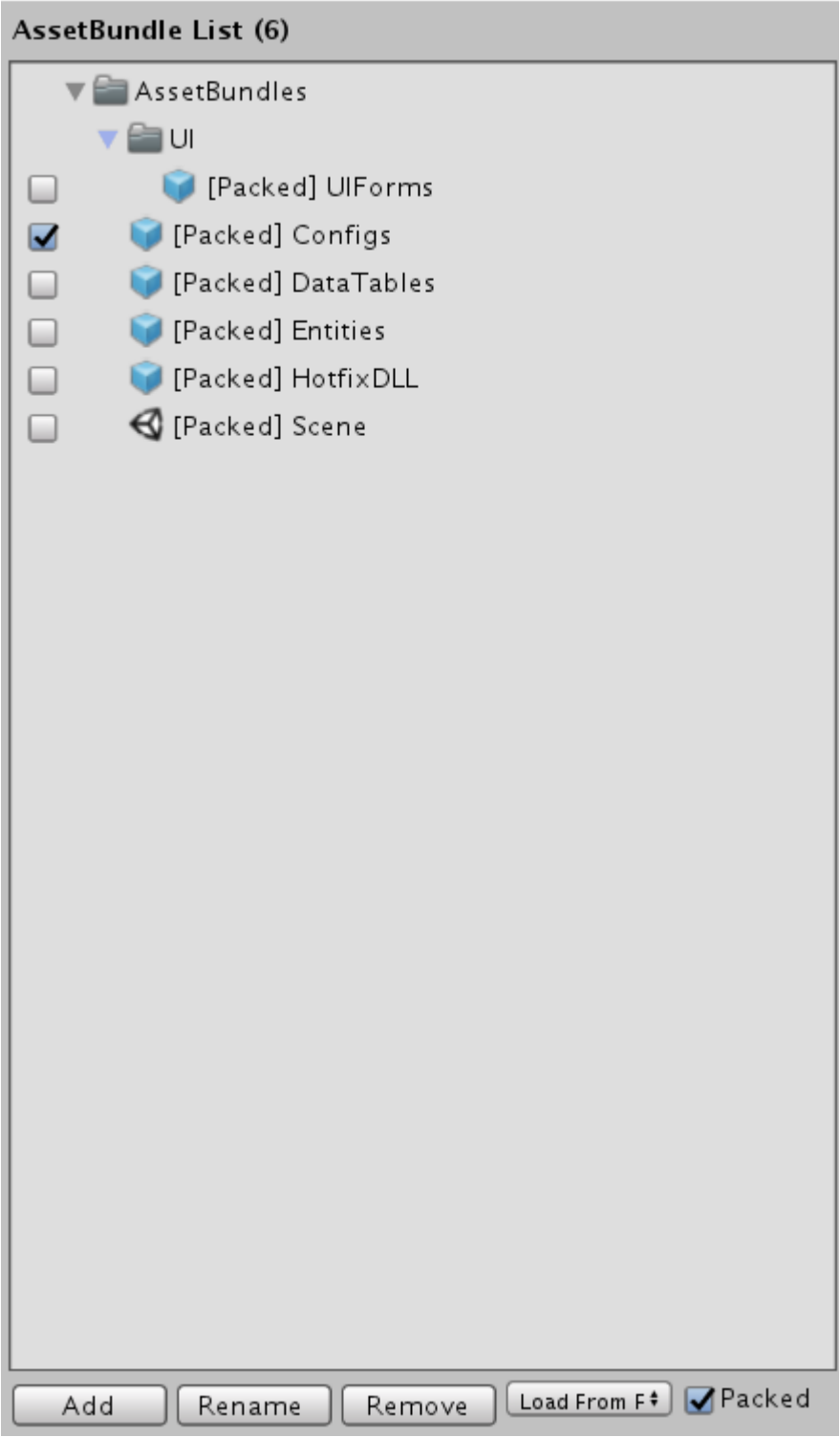
-  复制Hotfix.pdb到Assets/GameMain/HotfixDLL/ 完成
UnityEngine.Debug:Log(Object)
-  复制Hotfix.dll到Assets/GameMain/HotfixDLL/ 完成
UnityEngine.Debug:Log(Object)

热更新资源

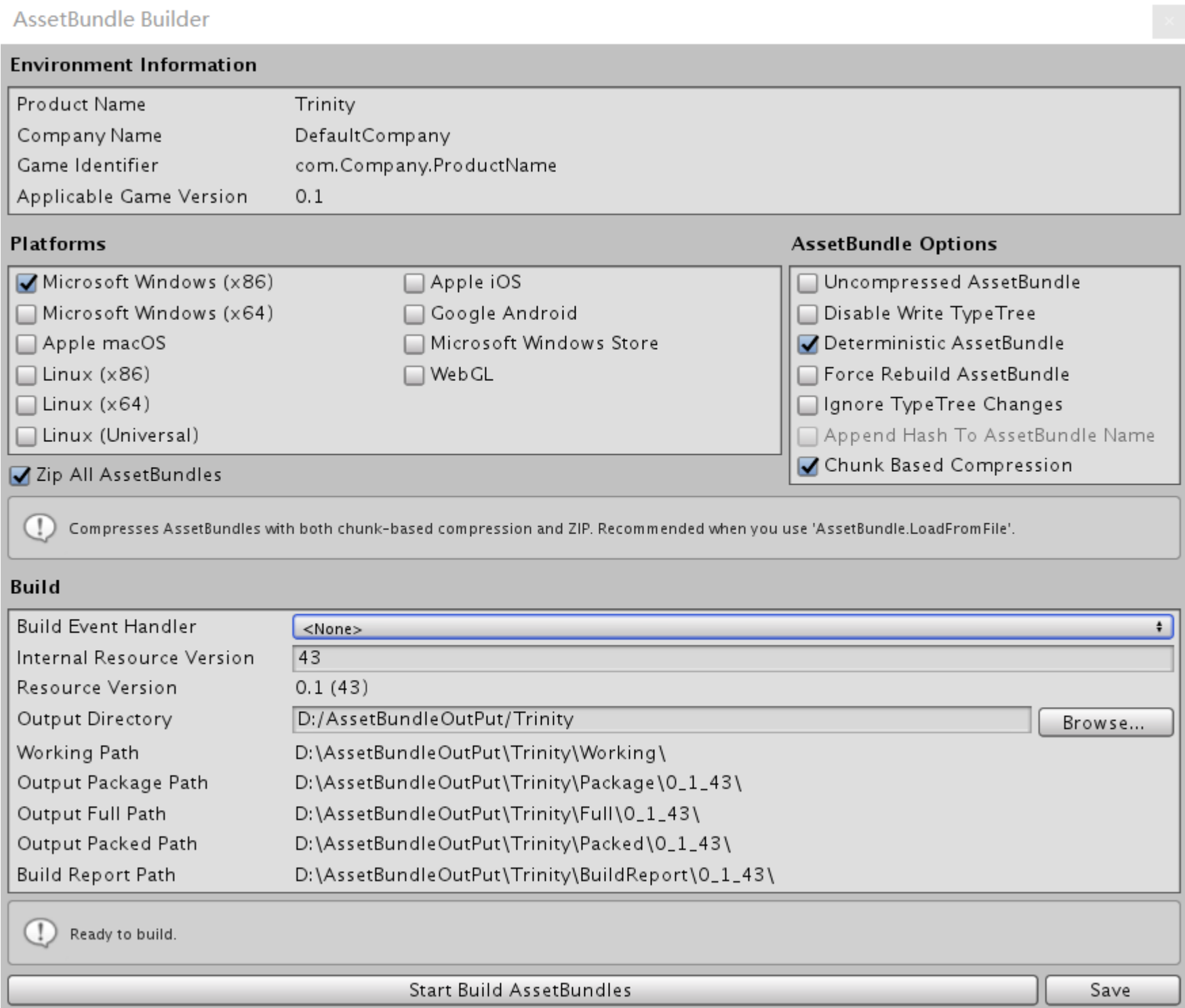
将Resource组件设置为Updatable模式



编辑AB包资源（需要勾选Packed）



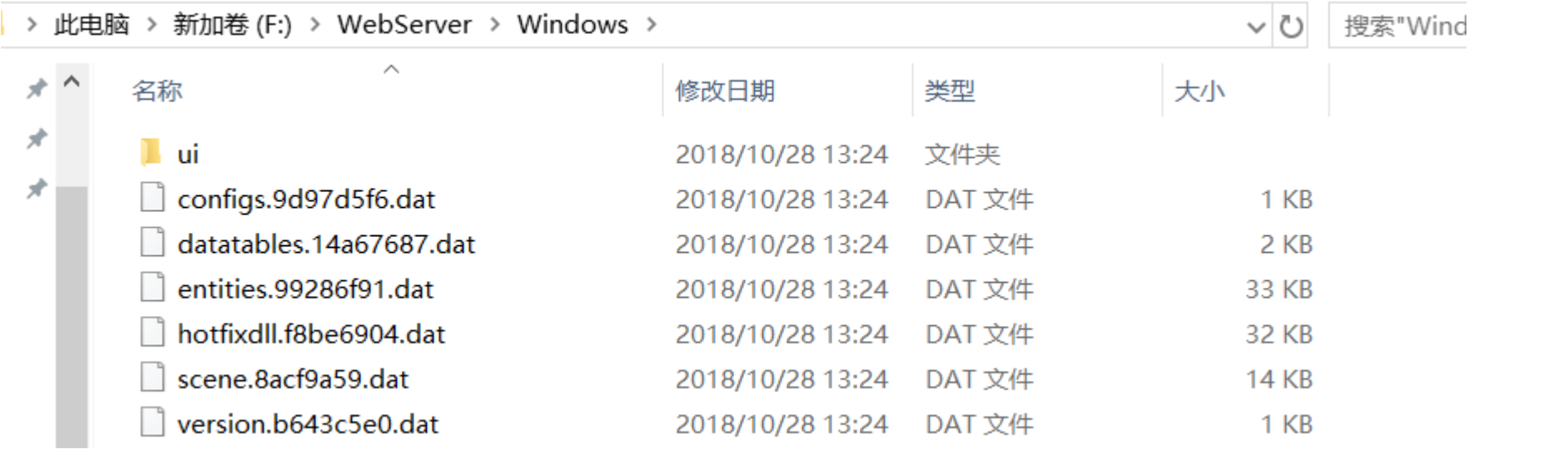
打出AB包（BuildEventHandler不要选择为自带的，否则会将单机模式下的资源复制到StreamingAssets下）



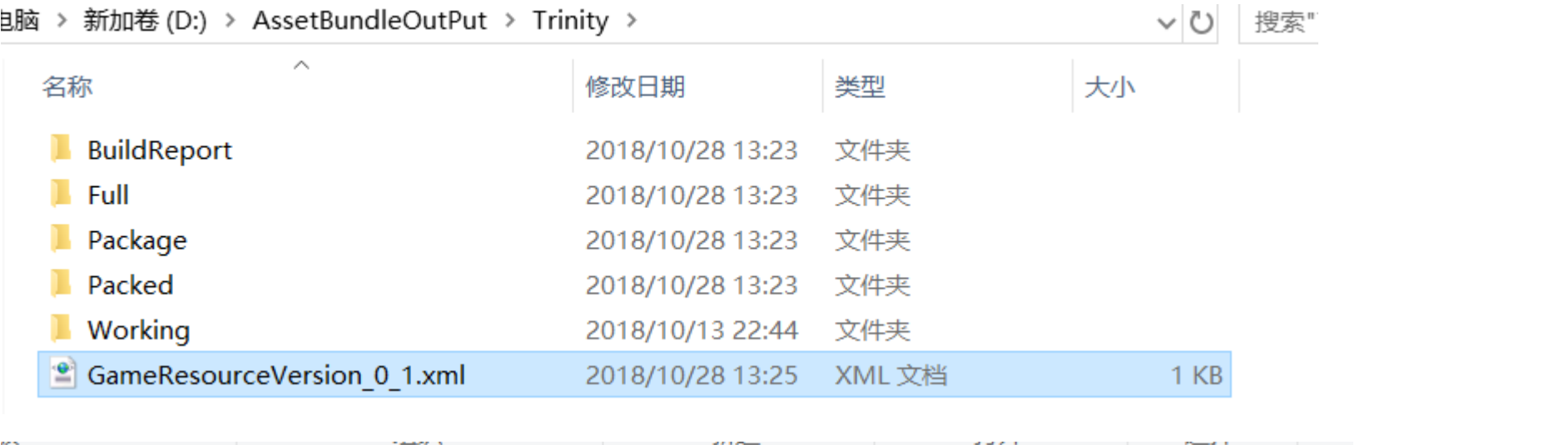
如果需要打包工程，则还需要将AB包输出目录的Packed目录下的资源复制到StreamingAssets下，作为打包时的资源



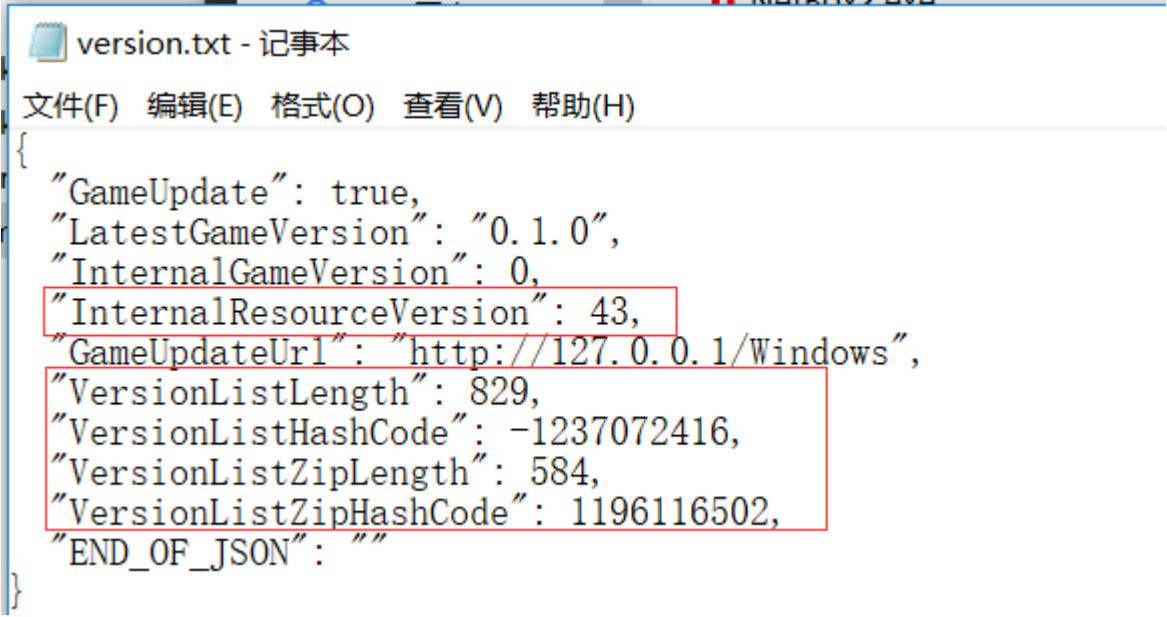
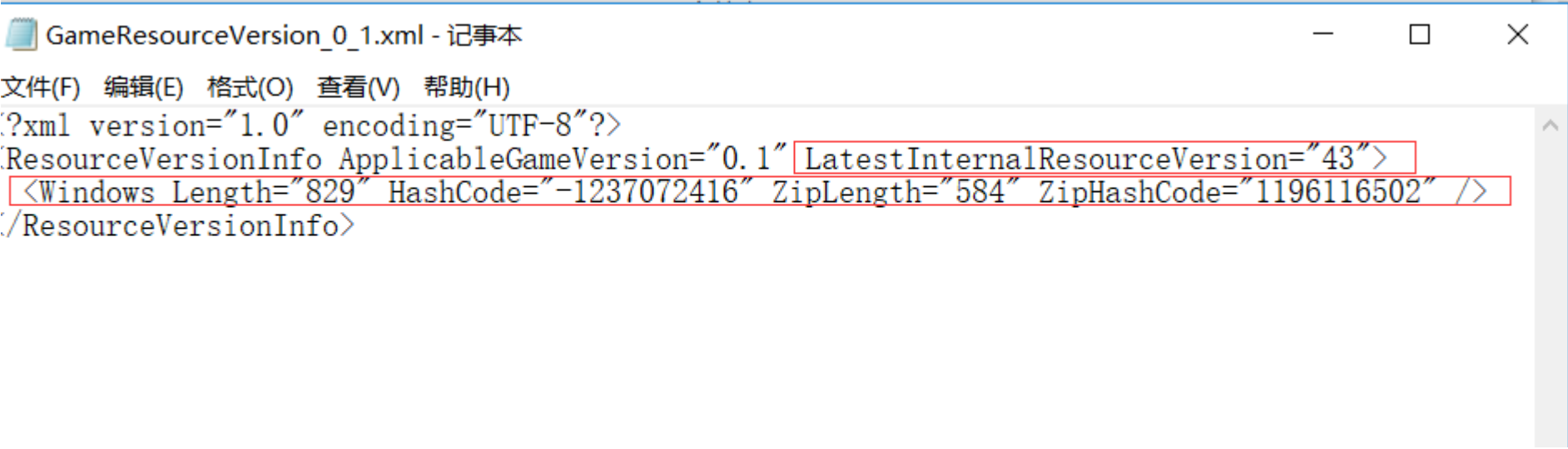
将AB包输出目录的Full目录下的资源复制到资源服务器上



然后根据AB包输出目录的GameResourceVersion文件来编写资源服务器上的Version.txt文件



主要根据GameResourceVersion红框部分的数据来对照着修改Version.txt中红框部分的数据



这样在游戏开始时便会进行资源更新的处理（要在编辑器下进行资源热更新的测试，需要将BaseComponent的EditorResourceMode取消勾选）

