# Unity ILRuntime + MVVM 架框热更新模块逻辑理解与整理

deepwaterooo

September 30, 2022

## Contents

# 1 Control：当写 Control 的时候感觉好像是 MVC, 但实际上仍然是 MVVM，后面会有热更新视图模块

## 1.1 Camera

### 1.1.1 CameraBase：相机抽象基类

```
// 相机抽象基类:  爱表哥，爱生活!!!
public abstract class CameraBase {

// 一堆相机操作相关的 setters/getters
    public Camera Camera {
        get;
        private set;
    }
    public GameObject GameObject {
        get;
        set;
    }
    public Transform Transform {
        get {
            return GameObject.transform;
        }
    }
    public virtual CameraWrapBase Wrap {
        get;
        protected set;
    }
    public Transform Target {
        get {
            return Wrap.Target;
        }
    }

    public CameraBase(GameObject go) {
        GameObject = go;
        Camera = GameObject.GetComponent<Camera>();
        Wrap = CreateWrap();
    }
    public void SetFieldOfView(float f) {
        if (Camera != null)
            Camera.fieldOfView = f;
    }

// 这些就可以实现与改写的呀
    public abstract CameraWrapBase CreateWrap();
    public virtual void OnTouchDrag(DragGesture gesture) {
        Wrap.OnTouchDrag(gesture);
    }
    public virtual void OnPinch(PinchGesture gesture) {
        Wrap.OnPinch(gesture);
    }
    protected virtual void Initialize() { }
}
```

### 1.1.2 Camera3D : CameraBase: 实体子类

```
// 3D 场景相机
public class Camera3D : CameraBase {
    public Camera3DWrap Camera3DWrap {
        get;
        set;
    }

    public Camera3D(GameObject go) : base(go) { }

    public override CameraWrapBase CreateWrap() {
        Camera3DWrap = GameObject.GetOrAddComponent<Camera3DWrap>();
        Wrap = Camera3DWrap;
        return Camera3DWrap;
    }
    public override void OnTouchDrag(DragGesture gesture) {
        // if (ViewManager.IsPo)
    }
}
```

## 1.2 Resource: 热更新里的资源处理桥接

### 1.2.1 abstract class ResourceHandleBase：服务器热更新资源包程序包处理：抽象基类

- 看见没有：Unity 的正常程序集里有一套，热更新程序集里也有一套；
- Unity 里是在公用接口类 IResourceLoader 里申请这些个公用方法；热更新里将所有相关方法申明在这个抽象基类里
- Unity 里 ResourceMap 是这个接口的唯一一个实现; Unity 里静态常量管理类 ResourceConstant 持有一个静态 ResourceMap 实例引用，取名叫 Loader
- 热更新里当前抽象基类 ResourceHandleBase 的继承子类 ResourceMapHandle 持有一个 IResourceLoader 公用接口的实例引用.
- 在游戏初始化的时候第一步做的事件是借助这个公用接口类，架起热更新程序集可以调用 Unity 里资源的通道桥梁，Unity 里的入口程序 GameApplication 里的初始化程序源码如下：

```
// 协程是说：游戏启动时，给这个控件 (gameObject) 加载运行时元件 ResourceMap (本质上是个程序脚本的实例化) ; 加载完毕自动触发
    IEnumerator Initialize() {

// 游戏初始时，实例化获取一个单例资源（管理类），架起两个不同程序集之间沟通的桥梁
        ResourceMap resourceMap = gameObject.AddComponent<ResourceMap>(); // 整个游戏应用，全局唯一
        resourceMap.OnInitializeSuccess += StartHotFix;  // 必备资源管理初始化好，自动触发游戏热更新程序模块集

// 将 Unity 程序集中初始化好的实例（通过公用接口）桥接 refer 给热更新程序集引用，从而实现热更新程序集可以调用 unity 中的资源
        ResourceConstant.Loader = resourceMap; // 可以重点看一下两个不同模块之间的资源管理的关系

        yield return new WaitForEndOfFrame();
    }
```

- 当前抽象基类类的源码如下：

```
// 服务器热更新资源包程序包处理：抽象基类
public abstract class ResourceHandleBase {

// 同步加载
#region Load
    public abstract T LoadAsset<T>(string bundleName, string assetName,
                                   EAssetBundleUnloadLevel unloadLevel =
                                   EAssetBundleUnloadLevel.ChangeSceneOver) where T : UnityEngine.Object;
    public abstract TMP_FontAsset LoadTMP_FontAsset(string bundleName, string assetName,
                                                    EAssetBundleUnloadLevel unloadLevel =
                                                    EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract Font LoadFont(string bundleName, string assetName,
                                  EAssetBundleUnloadLevel unloadLevel =
                                  EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract AnimationClip LoadAnimationClip(string bundleName, string assetName,
                                                    EAssetBundleUnloadLevel unloadLevel =
                                                    EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract AnimatorOverrideController LoadAnimatorOverrideController(string bundleName, string assetName,
                                                                              EAssetBundleUnloadLevel unloadLevel =
                                                                              EAssetBundleUnloadLevel.ChangeSceneOve
    public abstract RuntimeAnimatorController LoadRuntimeAnimatorController(string bundleName, string assetName,
                                                                            EAssetBundleUnloadLevel unloadLevel =
                                                                            EAssetBundleUnloadLevel.ChangeSceneOver)
    public abstract AudioClip LoadAudioClip(string bundleName, string assetName,
                                            EAssetBundleUnloadLevel unloadLevel =
                                            EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract Material LoadMaterial(string bundleName, string assetName,
                                          EAssetBundleUnloadLevel unloadLevel =
                                          EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract TextAsset LoadTextAsset(string bundleName, string assetName,
                                            EAssetBundleUnloadLevel unloadLevel =
                                            EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract Sprite LoadSprite(string bundleName, string assetName,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract Texture2D LoadTexture2D(string bundleName, string assetName,
                                            EAssetBundleUnloadLevel unloadLevel =
                                            EAssetBundleUnloadLevel.ChangeSceneOver);
    public abstract void LoadScene(string bundleName, string assetName,
                                   EAssetBundleUnloadLevel unloadLevel =
                                   EAssetBundleUnloadLevel.ChangeSceneOver, bool isAddtive = false);
    public abstract GameObject LoadClone(string bundleName, string assetName,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver);
#endregion

// 异步加载
#region LoadAsyn
    public abstract void LoadAssetAsyn<T>(string bundleName, string assetName, Action<T> onSuccess,
                                          EAssetBundleUnloadLevel unloadLevel =
                                          EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false
    public abstract void LoadTMP_FontAssetAsyn(string bundleName, string assetName, Action<TMP_FontAsset> onSuccess,
                                               EAssetBundleUnloadLevel unloadLevel =
                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = f
    public abstract void LoadFontAsyn(string bundleName, string assetName, Action<Font> onSuccess,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false);
    public abstract void LoadAnimationClipAsyn(string bundleName, string assetName, Action<AnimationClip> onSuccess,
                                               EAssetBundleUnloadLevel unloadLevel =
                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = f
    public abstract void LoadAnimatorOverrideControllerAsyn(string bundleName, string assetName, Action<AnimatorOver
                                                            EAssetBundleUnloadLevel unloadLevel =
                                                            EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInte
    public abstract void LoadRuntimeAnimatorControllerAsyn(string bundleName, string assetName, Action<RuntimeAnimato
                                                           EAssetBundleUnloadLevel unloadLevel =
                                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInte
    public abstract void LoadAudioClipAsyn(string bundleName, string assetName, Action<AudioClip> onSuccess,
                                           EAssetBundleUnloadLevel unloadLevel =
                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false
    public abstract void LoadMaterialAsyn(string bundleName, string assetName, Action<Material> onSuccess,
                                          EAssetBundleUnloadLevel unloadLevel =
                                          EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false
    public abstract void LoadTextAssetAsyn(string bundleName, string assetName, Action<TextAsset> onSuccess,
                                           EAssetBundleUnloadLevel unloadLevel =
                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false
    public abstract void LoadSpriteAsyn(string bundleName, string assetName, Action<Sprite> onSuccess,
                                        EAssetBundleUnloadLevel unloadLevel =
                                        EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false);
    public abstract void LoadTexture2DAsyn(string bundleName, string assetName, Action<Texture2D> onSuccess,
                                           EAssetBundleUnloadLevel unloadLevel =
                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false
    public abstract void LoadSceneAsyn(string bundleName, string assetName, Action onSuccess,
                                       EAssetBundleUnloadLevel unloadLevel =
                                       EAssetBundleUnloadLevel.ChangeSceneOver, bool isAddtive = false);
    public abstract void LoadCloneAsyn(string bundleName, string assetName, Action<GameObject> onSuccess,
                                       EAssetBundleUnloadLevel unloadLevel =
                                       EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false);
#endregion
#region Unload
    public abstract void Unload(string keyName, bool allObjects);
    public abstract void UnloadAll();
#endregion
    public abstract void LoadTexture2DAsyn(string name, Action<Texture2D> onSuccess, Action onFail, bool needCache);
}
```

### 1.2.2 ResourceHelper

```
// 资源加载接口类: 什么叫桥接?
public class ResourceHelper {
    static ResourceHandleBase _handle;
    static ResourceHandleBase Handle {
        get {
            if (_handle == null) {
                _handle = new ResourceMapHandle();
            }
            return _handle;
        }
    }
#region Load
    public static T LoadAsset<T>(string bundleName, string assetName,
                                 EAssetBundleUnloadLevel unloadLevel =
                                 EAssetBundleUnloadLevel.ChangeSceneOver) where T : UnityEngine.Object {
        return Handle.LoadAsset<T>(bundleName, assetName, unloadLevel);
    }
    public static TMP_FontAsset LoadTMP_FontAsset(string bundleName, string assetName,
                                                  EAssetBundleUnloadLevel unloadLevel =
                                                  EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadTMP_FontAsset(bundleName, assetName, unloadLevel);
    }
    public static Font LoadFont(string bundleName, string assetName,
                            EAssetBundleUnloadLevel unloadLevel =
                            EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadFont(bundleName, assetName, unloadLevel);
    }
    public static AnimationClip LoadAnimationClip(string bundleName, string assetName,
                                              EAssetBundleUnloadLevel unloadLevel =
                                              EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadAnimationClip(bundleName, assetName, unloadLevel);
    }
    public static AnimatorOverrideController LoadAnimatorOverrideController(string bundleName, string assetName,
                                                                        EAssetBundleUnloadLevel unloadLevel =
                                                                        EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadAnimatorOverrideController(bundleName, assetName, unloadLevel);
    }
    public static RuntimeAnimatorController LoadRuntimeAnimatorController(string bundleName, string assetName,
                                                                      EAssetBundleUnloadLevel unloadLevel =
                                                                      EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadRuntimeAnimatorController(bundleName, assetName, unloadLevel);
    }
    public static AudioClip LoadAudioClip(string bundleName, string assetName,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadAudioClip(bundleName, assetName, unloadLevel);
    }
    public static Material LoadMaterial(string bundleName, string assetName,
                                    EAssetBundleUnloadLevel unloadLevel =
                                    EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadMaterial(bundleName, assetName, unloadLevel);
    }
    public static TextAsset LoadTextAsset(string bundleName, string assetName,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadTextAsset(bundleName, assetName, unloadLevel);
    }
    public static Sprite LoadSprite(string bundleName, string assetName,
                                EAssetBundleUnloadLevel unloadLevel =
                                EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadSprite(bundleName, assetName, unloadLevel);
    }
    public static Texture2D LoadTexture2D(string bundleName, string assetName,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadTexture2D(bundleName, assetName, unloadLevel);
    }
    public static void LoadScene(string bundleName, string assetName,
                             EAssetBundleUnloadLevel unloadLevel =
                             EAssetBundleUnloadLevel.ChangeSceneOver, bool isAdditive = false) {
        Handle.LoadScene(bundleName, assetName, unloadLevel, isAdditive);
    }
    public static GameObject LoadClone(string bundleName, string assetName,
                                   EAssetBundleUnloadLevel unloadLevel =
                                   EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Handle.LoadClone(bundleName, assetName, unloadLevel);
    }
#endregion
#region LoadAsyn
    public static void LoadAssetAsyn<T>(string bundleName, string assetName, Action<T> onSuccess,
                                    EAssetBundleUnloadLevel unloadLevel =
                                    EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) where T
        Handle.LoadAssetAsyn<T>(bundleName, assetName, onSuccess, unloadLevel);
    }
    public static void LoadTMP_FontAssetAsyn(string bundleName, string assetName, Action<TMP_FontAsset> onSuccess,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadTMP_FontAssetAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadFontAsyn(string bundleName, string assetName, Action<Font> onSuccess,
                                EAssetBundleUnloadLevel unloadLevel =
                                EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadFontAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadAnimationClipAsyn(string bundleName, string assetName, Action<AnimationClip> onSuccess,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadAnimationClipAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadAnimatorOverrideControllerAsyn(string bundleName, string assetName, Action<AnimatorOverrideContr
                                                          EAssetBundleUnloadLevel unloadLevel =
                                                          EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoa
        Handle.LoadAnimatorOverrideControllerAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadRuntimeAnimatorControllerAsyn(string bundleName, string assetName, Action<RuntimeAnimatorControl
                                                         EAssetBundleUnloadLevel unloadLevel =
                                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad
        Handle.LoadRuntimeAnimatorControllerAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadAudioClipAsyn(string bundleName, string assetName, Action<AudioClip> onSuccess,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadAudioClipAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadMaterialAsyn(string bundleName, string assetName, Action<Material> onSuccess,
                                        EAssetBundleUnloadLevel unloadLevel =
                                        EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadMaterialAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadTextAssetAsyn(string bundleName, string assetName, Action<TextAsset> onSuccess,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadTextAssetAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
    public static void LoadSpriteAsyn(string bundleName, string assetName, Action<Sprite> onSuccess,
                                      EAssetBundleUnloadLevel unloadLevel =
                                      EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadSpriteAsyn(bundleName, assetName, onSuccess, unloadLevel);
    }
    public static void LoadTexture2DAsyn(string bundleName, string assetName, Action<Texture2D> onSuccess,
                                         EAssetBundleUnloadLevel unloadLevel =
                                         EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadTexture2DAsyn(bundleName, assetName, onSuccess, unloadLevel);
    }
    public static void LoadSceneAsyn(string bundleName, string assetName, Action onSuccess,
                                     EAssetBundleUnloadLevel unloadLevel =
                                     EAssetBundleUnloadLevel.ChangeSceneOver, bool isAdditive = false) {
        Handle.LoadSceneAsyn(bundleName, assetName, onSuccess, unloadLevel, isAdditive);
    }
    public static void LoadCloneAsyn(string bundleName, string assetName, Action<GameObject> onSuccess,
                                     EAssetBundleUnloadLevel unloadLevel =
                                     EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
        Handle.LoadCloneAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
    }
#endregion
#region Unload
#endregion
    public static void LoadTexture2DAsyn(string name, Action<Texture2D> onSuccess, Action onFail, bool needCache = true) {
        Handle.LoadTexture2DAsyn(name, onSuccess, onFail, needCache);
    }
}
```

### 1.2.3 ResourceMapHandle : ResourceHandleBase: 感觉这里两套程序集能够交接起来，这里需要好好想想理解清楚

```
public class ResourceMapHandle : ResourceHandleBase {
    IResourceLoader Loader {
        get {
            return ResourceConstant.Loader;
        }
    }
#region Load
    public override T LoadAsset<T>(string bundleName, string assetName,
                                   EAssetBundleUnloadLevel unloadLevel =
                                   EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Loader.LoadAsset<T>(bundleName, assetName, unloadLevel);
    }
    public override TMP_FontAsset LoadTMP_FontAsset(string bundleName, string assetName,
                                                    EAssetBundleUnloadLevel unloadLevel =
                                                    EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Loader.LoadTMP_FontAsset(bundleName, assetName, unloadLevel);
    }
    public override Font LoadFont(string bundleName, string assetName,
                                  EAssetBundleUnloadLevel unloadLevel =
                                  EAssetBundleUnloadLevel.ChangeSceneOver) {
        return Loader.LoadFont(bundleName, assetName, unloadLevel);
    }
```

```
        public override AnimationClip LoadAnimationClip(string bundleName, string assetName,
                                                        EAssetBundleUnloadLevel unloadLevel =
                                                        EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadAnimationClip(bundleName, assetName, unloadLevel);
        }
        public override AnimatorOverrideController LoadAnimatorOverrideController(string bundleName, string assetName,
                                                                                  EAssetBundleUnloadLevel unloadLevel =
                                                                                  EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadAnimatorOverrideController(bundleName, assetName, unloadLevel);
        }
        public override RuntimeAnimatorController LoadRuntimeAnimatorController(string bundleName, string assetName,
                                                                                EAssetBundleUnloadLevel unloadLevel =
                                                                                EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadRuntimeAnimatorController(bundleName, assetName, unloadLevel);
        }
        public override AudioClip LoadAudioClip(string bundleName, string assetName,
                                                EAssetBundleUnloadLevel unloadLevel =
                                                EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadAudioClip(bundleName, assetName, unloadLevel);
        }
        public override Material LoadMaterial(string bundleName, string assetName,
                                              EAssetBundleUnloadLevel unloadLevel =
                                              EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadMaterial(bundleName, assetName, unloadLevel);
        }
        public override TextAsset LoadTextAsset(string bundleName, string assetName,
                                                EAssetBundleUnloadLevel unloadLevel =
                                                EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadTextAsset(bundleName, assetName, unloadLevel);
        }
        public override Sprite LoadSprite(string bundleName, string assetName,
                                          EAssetBundleUnloadLevel unloadLevel =
                                          EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadSprite(bundleName, assetName, unloadLevel);
        }
        public override Texture2D LoadTexture2D(string bundleName, string assetName,
                                                EAssetBundleUnloadLevel unloadLevel =
                                                EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadTexture2D(bundleName, assetName, unloadLevel);
        }
        public override void LoadScene(string bundleName, string assetName,
                                       EAssetBundleUnloadLevel unloadLevel =
                                       EAssetBundleUnloadLevel.ChangeSceneOver, bool isAddtive = false) {
            Loader.LoadScene(bundleName, assetName, unloadLevel);
        }
        public override GameObject LoadClone(string bundleName, string assetName,
                                             EAssetBundleUnloadLevel unloadLevel =
                                             EAssetBundleUnloadLevel.ChangeSceneOver) {
            return Loader.LoadClone(bundleName, assetName, unloadLevel);
        }
    #endregion
    #region LoadAsyn
        public override void LoadAssetAsyn<T>(string bundleName, string assetName, Action<T> onSuccess,
                                              EAssetBundleUnloadLevel unloadLevel =
                                              EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadAssetAsyn<T>(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadTMP_FontAssetAsyn(string bundleName, string assetName, Action<TMP_FontAsset> onSuccess,
                                                   EAssetBundleUnloadLevel unloadLevel =
                                                   EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false)
            Loader.LoadTMP_FontAssetAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadFontAsyn(string bundleName, string assetName, Action<Font> onSuccess,
                                          EAssetBundleUnloadLevel unloadLevel =
                                          EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadFontAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadAnimationClipAsyn(string bundleName, string assetName, Action<AnimationClip> onSuccess,
                                                   EAssetBundleUnloadLevel unloadLevel =
                                                   EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false)
            Loader.LoadAnimationClipAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadAnimatorOverrideControllerAsyn(string bundleName, string assetName, Action<AnimatorOverrideCon
                                                                EAssetBundleUnloadLevel unloadLevel =
                                                                EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptL
            Loader.LoadAnimatorOverrideControllerAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadRuntimeAnimatorControllerAsyn(string bundleName, string assetName, Action<RuntimeAnimatorContr
                                                               EAssetBundleUnloadLevel unloadLevel =
                                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLo
            Loader.LoadRuntimeAnimatorControllerAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadAudioClipAsyn(string bundleName, string assetName, Action<AudioClip> onSuccess,
                                               EAssetBundleUnloadLevel unloadLevel =
                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadAudioClipAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadMaterialAsyn(string bundleName, string assetName, Action<Material> onSuccess,
                                              EAssetBundleUnloadLevel unloadLevel =
                                              EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadMaterialAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadTextAssetAsyn(string bundleName, string assetName, Action<TextAsset> onSuccess,
                                               EAssetBundleUnloadLevel unloadLevel =
                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadTextAssetAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadSpriteAsyn(string bundleName, string assetName, Action<Sprite> onSuccess,
                                            EAssetBundleUnloadLevel unloadLevel =
                                            EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadSpriteAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadTexture2DAsyn(string bundleName, string assetName, Action<Texture2D> onSuccess,
                                               EAssetBundleUnloadLevel unloadLevel =
                                               EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadTexture2DAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
        public override void LoadSceneAsyn(string bundleName, string assetName, Action onSuccess,
                                           EAssetBundleUnloadLevel unloadLevel =
                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isAddtive = false) {
            Loader.LoadSceneAsyn(bundleName, assetName, onSuccess, unloadLevel, isAddtive);
        }
        public override void LoadCloneAsyn(string bundleName, string assetName, Action<GameObject> onSuccess,
                                           EAssetBundleUnloadLevel unloadLevel =
                                           EAssetBundleUnloadLevel.ChangeSceneOver, bool isForceInterruptLoad = false) {
            Loader.LoadCloneAsyn(bundleName, assetName, onSuccess, unloadLevel, isForceInterruptLoad);
        }
    #endregion
    #region Unload
        public override void Unload(string keyName, bool allObjects) {
            Loader.Unload(keyName, allObjects);
        }
        public override void UnloadAll() {
            Loader.UnloadAll();
        }
    #endregion
        public override void LoadTexture2DAsyn(string name, Action<Texture2D> onSuccess, Action onFail, bool needCache) {
            Loader.LoadTexture2DAsyn(name, onSuccess, onFail, needCache);
        }
    }
```

## 1.3 Scene: 场景热更新反序列化为 unity 游戏引擎所能识别和可执行场景控件程序等

- 回想一下热更新的过程：从网络服务器上下载出需要的资源包或是程序包，它是序列化后的数据；要把它还原成 unity 认得认识的场景等，还需要一个反序化的过程
- 那么这个包裹里，就专职负责：如何从下载的资源包里游戏引擎场景相关的序列化数据还原为 unity 游戏引擎中认识并且可执行的游戏引擎程序集里的场景控件等，这个过程，所有场景相关的
- 那么这里可以封装和包含的几个类就包括了：如果可以封装成为的（抽象）基类；根据各个不同应用实际需要而可以实例化成了几个不同的场景类型的反序列化定义类；如果可以，如果有好几个不同的场景，如有必要，还可以有一个必要的管理多个不同场景的场景管理者类

### 1.3.1 abstract class SceneBase：热更新场景抽象基类

```
// 热更新场景基类：
public abstract class SceneBase {
    public SceneData Data {
        get;
        set;
    }
    public SceneTypeData TypeData {
        get;
        set;
    }
    public GameObject GameObject {
        get;
        set;
    }
    // 家具的集合
    public Dictionary<int, FurnitureBase> furnitures = new Dictionary<int, FurnitureBase>();
    public SceneBase(int type) {
        Data = new SceneData();
        Data.type = type;
        Data.materialDatas = new Dictionary<string, MaterialData>();
        Data.furnitureDatas = new Dictionary<int, FurnitureData>();
        TypeData = TypeDataManager.GetSceneTypeData(Data.type);
    }
    public SceneBase(SceneData data) {
        Data = data;
        TypeData = TypeDataManager.GetSceneTypeData(Data.type);
    }
    public void LoadSceneGameObject() {
        ResourceConstant.Loader.LoadCloneAsyn(TypeData.bundleName, TypeData.assetName, (go) => {
            GameObject = go;
            SetGameObjectName();
            SceneManager.Instance.CurrentScene = this;
            Initialize();
        }, EAssetBundleUnloadLevel.ChangeSceneOver);
    }
    protected abstract void SetGameObjectName();
    protected abstract void Initialize();
    public virtual void Dispose() {
        ResourceConstant.Loader.Unload(TypeData.bundleName, true);
    }
}
```

### 1.3.2 热更新用到三种不同场景中的一个样例，最简单最特殊的 ShowScene

```
// 空间秀场景: 吃货天空的鬼窝秀
public class ShowScene : SceneBase {
    // 这些这里可以有 unity 认得的至少一部分控件 (因为热更新程序持有 UnityEngine.dll 的程序集引用？想一想)
    public Transform OneGameObjectRoot { // Transform Transform ..... 某个游戏控件的位置信息
        get;
        set;
    }
    public ShowScene(int type) : base(type) {}
    public ShowScene(SceneData data) : base(data) {}

    protected override void SetGameObjectName() {
        GameObject.name = "Show_" + TypeData.id;
    }
    protected override void Initialize() {
        OneGameObjectRoot = GameObject.FindChildByName("OneGameObjectRoot").transform;
        InitializeTypetwos();
    }
    void InitializeTypetwos() { // 初始化某种类型 Typetwo 类型的元件 (控件或集)
        foreach (var typetwoData in Data.typetwoDatas.Values) {
            TypetwoTypeData typetwoTypeData = TypeDataManager.GetTypetwoTypeData(typetwoData.type);
            TypetwoBase typetwoBase;
            // 再然后, 可以根据 Typetwo 类型是单个数据还是集合类再作进一步的初始化到最小元件单位 (? 就是层层折解, 反序列化实例化成真
            // 这里的部分逻辑略去
        }
    }
    public override void Dispose() {
        base.Dispose();
        foreach (var typetwo in typetwos.Values) {
            typetwo.Dispose();
        }
    }
}
```

### 1.3.3 SceneManager: 场景管理器, 对于有需要必要时需要进行热更新的场景的相关逻辑 (加载，创建，渐近切换等) 的封装定义管理

```
// 场景管理器: 我的游戏里也写过好几个不同的管理类, 热更新里的管理类与 unity 里的普通管理类有什么不同呢?
public class SceneManager { // unity 里不同场景的编号不同,  这里以 int 值不同来区分不同的游戏场景

    private static SceneManager instance;  // 希望是单例模式, 不涉及多线程安全
    public static SceneManager Instance {
        get {
            if (instance == null)
                instance = new SceneManager();
            return instance;
        }
    }

    // 当前场景
    public SceneBase CurrentScene {
        get;
        set;
    }
    public int currentSelectGameObjectInstanceID;

    // 创建一个新场景: 根据传进来的参数值, 来轮询实例化对象的场景 (这个包裹里不是也定义了各个不同场景的继承于抽象基类的继承类的反序列
    public void CreateNewScene(int type) { // unity 里不同场景的编号不同,  这里以 int 值不同来区分不同的游戏场景
        ClearLastSceneGameObject();
        currentSelectGameObjectInstanceID = 100000001;
        SceneTypeData typeData = TypeDataManager.GetSceneTypeData(type);
        if (typeData.type == (int)ESceneType.Edit) {
            CurrentScene = new EditScene(type);
            CurrentScene.LoadSceneGameObject();
        } else if (typeData.type == (int)ESceneType.Show) {
            CurrentScene = new ShowScene(type);
            CurrentScene.LoadSceneGameObject();
        } else if (typeData.type == (int)ESceneType.Camera) {
            CurrentScene = new CameraScene(type);
            CurrentScene.LoadSceneGameObject();
        }
    }

    // 加载一个场景
    public void LoadScene(SceneData data) {
        ClearLastSceneGameObject();
        currentSelectGameObjectInstanceID = data.GetMaxFurnitureInstanceID();
        SceneTypeData typeData = TypeDataManager.GetSceneTypeData(data.type);
        if (typeData.type == (int)ESceneType.Edit) {
            CurrentScene = new EditScene(data);
            CurrentScene.LoadSceneGameObject();
        } else if (typeData.type == (int)ESceneType.Show) {
            CurrentScene = new ShowScene(data);
            CurrentScene.LoadSceneGameObject();
        } else if (typeData.type == (int)ESceneType.Camera) {
            CurrentScene = new CameraScene(data);
            CurrentScene.LoadSceneGameObject();
        }
    }
    void ClearLastSceneGameObject() {
        if (CurrentScene != null && CurrentScene.GameObject != null)
            Object.DestroyImmediate(CurrentScene.GameObject);
    }
    public void CleanCurrentScene() {
        if (CurrentScene != null && CurrentScene.GameObject != null) {
            Object.DestroyImmediate(CurrentScene.GameObject);
            CurrentScene.Dispose();
            CurrentScene = null;
            Resources.UnloadUnusedAssets();
            System.GC.Collect();
        }
    }
    // 加载某个特殊的场景: 那么就是说对于某类过于特殊的场景, 可以将其单独打包成一个资源包上传服务器和从服务器下载下来加载
    // 这里这个特殊的场景, 好像不涉及任何其它数据 (除了一个应用里的 SceneData)? 所以过程极为简单, 可以跳过很多步
    // 再想一下: 这个特殊的场景到底特殊在哪里, 可以独立有条短路的加载方法?
    public void LoadShowScene(string bundleName, string assetName) {
        string json = ResourceConstant.Loader.LoadTextAsset(bundleName, assetName, EAssetBundleUnloadLevel.LoadOver).text;
        //Debug.Log("json: " + json);
        SceneData sceneData = SceneData.JsonToObject(json);
        LoadScene(sceneData);
    }
}
```

# 2 Data: 数据驱动

- **这里的问题**：同样是为了打包和折包场景，为什么一个场景数据要定义成两种不同的场景类，相当于普通数据类型的装箱与折箱操作么，为了自适应两个不同的程序集？序列化与反序列化？要好好想一想
- 前面有对 ViewModel 的适配包装，这里更多的是 frameworks 层如同 SquarePanda 里有专用的公司开发的 SDK 来对这同一家公司里的 10 款游戏进行通用模块包装一样，这个游戏热更新框架可以理解为是对同一家或是几家相同架构设计的多款游戏的架构封装，那么各个游戏独特的数据类型 model 仍然该是封装在各自游戏应用的内部；framework 热更新框架只提取各个游戏通用的逻辑进行封装。

也提到说数据类型是

- 热更新所有的一切（资源包和程序包）都打包成数据上传热更新服务器，所以是数据驱动，一切都是数据!!
- 在把不同的所有需要热更新的东西全部打包上传服务器的过程中，背后本质是不同类型打包基类元件（资源包，场景等）的序列化（上传服务器）与反序列化（下载资源包后的加载过程?）
- 那么如何把不同的热更新类型打包封装成不同的数据呢（不同数据类型的自定义）

## 2.1 Data: 必要的序列化过程中自己封装成的自定义类型仍然可以或是需要定义在这里

- 我自己的游戏里写过俄罗斯广场砖的序列化与反序列化，可是我还不曾真正去读自己几年前写过的源码，忘得差不多了，该是时候把它们捡起来再深入发展一下了（爱表哥，爱生活!!!）
- 可以养成一个编程习惯就是说：凡是带 Data 作为后缀的，都当作定义为序列化数据的方法逻辑？

### 2.1.1 SceneData：场景实例数据

```
// 场景实例数据
public class SceneData { // 再去多想一想，为什么要用 SceneData 与 SceneTypeData 相区分？
    // 实例 ID
    public int instanceID;
    // 场景类型
    public int type;

// 这是同一家公司或是相同架构公司多个不同游戏的 framework 游戏框架通用管理
    // 接下来：需要对各自游戏应用各不同场景下的 Model 数据进行集装，封装到自定义的场景热更自定义类型里去
    // 可是用集合类对同一场景下的同一数据类型进行集装；
    // 同一场景下不同数据类型间，可以生成多个不同字典等集合数据结构进行集装
    // 单个数据
    public SingleData singleData;
    // 某数据类型的集合
    public Dictionary<string, TypeoneData> typeoneDatas;
    // 另一数据类型的集合
    public Dictionary<int, TypetwoData> typetwoDatas;

    // 反序列化
    public static SceneData JsonToObject(string json) {
        SceneData data = new SceneData();
        JsonObject jsonObject = JsonSerializer.Deserialize(json) as JsonObject;
        if (jsonObject != null) {
            if (jsonObject.ContainsKey("instanceID")) {
                data.instanceID = jsonObject["instanceID"];
            }
            data.type = jsonObject["type"];
            if (jsonObject.ContainsKey("singleData")) {
                JsonValue singleJonValue = jsonObject["singleData"];
                data.singleData = SingleData.JsonToObject(singleJonValue.ToString());
            }
            data.typeoneDatas = new Dictionary<string, TypeoneData>();
            JsonValue jsonValue = jsonObject["typeoneDatas"];
            JsonArray jsonArray = JsonSerializer.Deserialize(jsonValue.ToString()) as JsonArray;
            foreach (var value in jsonArray) {
                TypeoneData typeoneData = TypeoneData.JsonToObject(value.ToString());
                data.typeoneDatas.Add(typeoneData.gameObjectName, typeoneData);
            }
            data.typetwoDatas = new Dictionary<int, TypetwoData>();
            JsonValue jsonValue2 = jsonObject["typetwoDatas"];
            JsonArray jsonArray2 = JsonSerializer.Deserialize(jsonValue2.ToString()) as JsonArray;
            foreach (var value in jsonArray2) {
                TypetwoData typetwoData = TypetwoData.JsonToObject(value.ToString());
                data.typetwoDatas.Add(typetwoData.instanceID, typetwoData);
            }
        }
        return data;
    }
    public override string ToString() {
        return ObjectToJson().ToString();
    }
    // 序列化
    public JsonObject ObjectToJson() {
        JsonObject jsonObject = new JsonObject();
        jsonObject.Add("instanceID", instanceID);
        jsonObject.Add("type", type);
        JsonObject singleJsonObject = singleData.ObjectToJson();
        jsonObject.Add("singleData", singleJsonObject);
        JsonArray jsonArray = new JsonArray();
        foreach (var data in typeoneDatas.Values) {
            JsonObject dataJsonObject = data.ObjectToJson();
            jsonArray.Add(dataJsonObject);
        }
        jsonObject.Add("typeoneDatas", jsonArray.ToString());
        JsonArray jsonArray2 = new JsonArray();
        foreach (var data in typetwoDatas.Values) {
            JsonObject dataJsonObject = data.ObjectToJson();
            jsonArray2.Add(dataJsonObject);
        }
        jsonObject.Add("typetwoDatas", jsonArray2.ToString());
        return jsonObject;
    }
    public int GetMaxTypetwoInstanceID() {
        int maxID = 100000001;
        foreach (var key in typetwoDatas.Keys) {
            if (key > maxID) {
                maxID = key;
            }
        }
        return maxID;
    }
}
```

## 2.2 TypeData

### 2.2.1 SceneTypeData: 场景类型数据

```
// 这里可以自定义几个不同的场景类型，方便程序区分
public enum ESceneType {
    None = 0,
    Edit = 1,
    Show = 2,
    Camera = 3
}
// 场景类型数据
public class SceneTypeData {
    public long id;
    public string gameObjectName;
    public string name;
    public string description;

    public int type;
    public string bundleName;
    public string assetName;
    public string iconBundleName;
    public string iconAssetName;

    // 把序列化数据重新反序列化成 unity 场景数据
    public static SceneTypeData JsonToObject(string json) {
        SceneTypeData typeData = new SceneTypeData();
        JsonObject jsonObject = JsonSerializer.Deserialize(json) as JsonObject;
        if (jsonObject != null) {
            typeData.id = jsonObject["id"];
            typeData.gameObjectName = jsonObject["gameObjectName"].ToString();
            typeData.name = jsonObject["name"].ToString();
            typeData.description = jsonObject["description"].ToString();
            typeData.type = jsonObject["type"];
            typeData.bundleName = jsonObject["bundleName"].ToString();
            typeData.assetName = jsonObject["assetName"].ToString();
            typeData.iconBundleName = jsonObject["iconBundleName"].ToString();
            typeData.iconAssetName = jsonObject["iconAssetName"].ToString();
        }
        return typeData;
    }
    public override string ToString() {
        return "id: " + id + " gameObjectName: " + gameObjectName + " name: " + name + " description: " + description
            + " type: " + type + " bundleName: " + bundleName + " assetName: " + assetName
            + " iconBundleName: " + iconBundleName + " iconAssetName: " + iconAssetName;
    }
}
```

## 2.3 static 静态管理类 TypeDataManager：文件比较大，仅以场景数据一种自定义数据类型来作分析

```
// TypeData 管理器: 静态管理类
public static class TypeDataManager { // 文件比较大，仅以场景数据一种类型来作分析

#region TypeDatas
    // 对于每种自定义自封装的类型，启用一个字典来进行管理；同自定义类型的数据，用一个长量型的 long 作为 key 来进行区分实例
    static Dictionary<long, SceneTypeData> sceneTypeDatas;

    public static Dictionary<long, SceneTypeData> GetSceneTypeDatas() {
        return sceneTypeDatas;
    }
    public static SceneTypeData GetSceneTypeData(long id) {
        if (sceneTypeDatas.ContainsKey(id)) {
            return sceneTypeDatas[id];
        } else {
            return null;
        }
    }
#endregion
    // 热更新起始时，资源包里：对于不同场景的初始化
    public static void InitializeTypeDatas() { // 那么加载的是场景（场景专用资源包吗？是的；不是资源包里关于场景的那一小部分）场
        string sceneJson = ResourceHelper.LoadTextAsset("scene/config/scene", "scene", EAssetBundleUnloadLevel.LoadOver).te
        if (!string.IsNullOrEmpty(sceneJson)) // 只要从场景资源包里的读出的字符串非空，就反序列化成特定类型备用
            InitializeSceneTypeData(sceneJson);
    }
    static void InitializeSceneTypeData(string jsonStr) { // 反序列化，将序列化数据反转成通用场景数据
        if (jsonStr != null) {
            sceneTypeDatas = new Dictionary<long, SceneTypeData>();
            JsonArray jsonArray = JsonSerializer.Deserialize(jsonStr) as JsonArray;
            if (jsonArray != null) {
                foreach (JsonValue jsonValue in jsonArray) {
                    SceneTypeData typeData = SceneTypeData.JsonToObject(jsonValue.ToString());
                    if (!sceneTypeDatas.ContainsKey(typeData.id)) // 当前资源管理器还没有这种类型 (int id) 的场景呢，就添加上，
                        sceneTypeDatas.Add(typeData.id, typeData);
                    else
                        Debug.LogError("sceneTypeDatas contains key: " + typeData.id);
                }
            } else
                Debug.LogError("sceneTypeData jsonArray is null");
        }
    }
}
```

- 这里面源码里面关于资源包里如何加载场景还弄得不清不楚，那么按照这里代码的提示，找出一个资源包，还看看它是怎么序列化场景数据的
- 关于当前资源包里场景的部分，是存在一个名叫 scene.txt 的文件里

```
[
  {
    "id": 10001.0,
    "gameObjectName": "show_10001",
    "name": "me 的小房间",
    "description": " 这是 me 的小房间 10001",
    "type": 2.0,
    "bundleName": "scene/scene/show/show_10001",
    "assetName": "show_10001",
    "iconBundleName": "scene/sceneicon/show/show_10001",
    "iconAssetName": "show_10001"
  },
  {
    "id": 10002.0,
    "gameObjectName": "edit_10002",
    "name": " 表哥房",
    "description": " 这是表哥现房 10002",
    "type": 1.0,
    "bundleName": "scene/scene/edit/edit_10002",
    "assetName": "edit_10002",
    "iconBundleName": "scene/sceneicon/edit/edit_10002",
    "iconAssetName": "edit_10002"
  }
]
```

## 3 UI: 热更新视图的相关逻辑

- 这里首先搞清楚一个常识概念就是：对于游戏里的不同场景，不知道是否可以实现同一个应用里的不同场景切换，但至少是可以保证一个应用里至少可以顺利加载一个场景的吧
- 上面说的是游戏里的场景；但是这里用的是 UI，是指游戏某一个场景里的游戏引擎场景（Scene）不变的情况下，只切换变化某一个或是几个 Panel，而把这一个又一个不同的 panel 当作是我先前写安卓 MVVM 设计里的不同 screen 屏幕；每个 Panel(View) 在这种框架设计里也会自动绑定跨域继承的 ViewModelBase 的继承实体类（从而实现视图与视图模型的自动绑定），从而实现在 Panel 级别的 MVVM 数据绑定设计模式。

### 3.1 TODO: (删减一下!!! ) 某个热更新视图 (这里是抽个例子出来参考一下)

```
// 热更新里拿取一个样例热更新的视图：这种视图热更新在网上应该能够成堆地找到
public class TestView : UnityGuiView {
    Button buttonOne;
    Button buttonTwo;
    Button buttonThree;
    GameObject pullDownRefresh;
    TextMeshProUGUI refreshText;
    DelayLoadGrid findItemGridRoot;
    ScrollViewEvent scrollViewEvent;
    List<GridItem> findGridItems = new List<GridItem>();
    List<FindData> findDatas;

    protected override void OnInitialize() {
        base.OnInitialize();  //  对基类抽象方法的继承实现
        buttonOne = GameObject.FindChildByName("ButtonOne").GetComponent<Button>();
        buttonOne.onClick.AddListener(OnClickButtonOne);
        buttonTwo = GameObject.FindChildByName("ButtonTwo").GetComponent<Button>();
        buttonTwo.onClick.AddListener(OnClickButtonTwo);
        buttonThree = GameObject.FindChildByName("ButtonThree").GetComponent<Button>();
        buttonThree.onClick.AddListener(OnClickButtonThree);
        findItemGridRoot = GameObject.FindChildByName("FindItemGridRoot").GetComponent<DelayLoadGrid>();
        scrollViewEvent = GameObject.FindChildByName("FindItemGridScrollView").GetComponent<ScrollViewEvent>();
        scrollViewEvent.onBeginDrag = OnBeginDrag;
        scrollViewEvent.onDraging = OnDraging;
        scrollViewEvent.onEndDrag = OnEndDrag;
        pullDownRefresh = GameObject.FindChildByName("PulldownRefresh");
        refreshText = pullDownRefresh.FindChildByName("Text").GetComponent<TextMeshProUGUI>();
        pullDownRefresh.SetActive(false);
        InitializeDatas(); // <<<<<<<<<<<<<<<<<<<<
    }
    public override void OnAppear() {
        base.OnAppear();
        CloseOtherRootView = CloseOtherRootViews;
    }
    void CloseOtherRootViews() {
        ViewManager.CloseOtherRootViews(ViewName);
    }
    void InitializeDatas() {
        findDatas = new List<FindData>();
        for (int i = 0; i < 10000; i++) {
            FindData findData;
            if (i % 7 == 0) {
                findData = new FindData(i, i.ToString(), 250);
            } else if (i % 7 == 1) {
                findData = new FindData(i, i.ToString(), 300);
            } else if (i % 7 == 2) {
                findData = new FindData(i, i.ToString(), 220);
            } else if (i % 7 == 3) {
                findData = new FindData(i, i.ToString(), 180);
            } else if (i % 7 == 4) {
                findData = new FindData(i, i.ToString(), 280);
            } else if (i % 7 == 5) {
                findData = new FindData(i, i.ToString(), 320);
            } else {
                findData = new FindData(i, i.ToString());
            }
            findDatas.Add(findData);
        }
    }
    // Enter2D
    void OnClickButtonOne() {
        TestCreateCustomEditScene.Instance.CreateCustomEditScene();
    }
    void OnClickButtonTwo() {
        SceneManager.Instance.LoadSpaceShowScene("scene/config/spaceshowscenedata/spaceshow_10001", "SpaceShow_10001");
    }
    void OnClickButtonThree() {
        findItemGridRoot.gameObject.DestoryImmediateAllChildren();
        findGridItems.Clear();
        if (findDatas != null && findDatas.Count > 0) {
            foreach (var findData in findDatas) {
                GridItem gridItem = new GridItem();
                gridItem.Name = "FindItemPrefab_" + findData.id;
                findGridItems.Add(gridItem);
                FindItemTemp findItem = new FindItemTemp(gridItem, findData);
            }
        }
        findItemGridRoot.InitializeItems(findGridItems);
    }
    void OnBeginDrag() {
        Debug.Log("OnBeginDrag");
    }
    void OnDraging() {
        if (findItemGridRoot.GetComponent<RectTransform>().anchoredPosition.y < 0f) {
            pullDownRefresh.SetActive(true);
            refreshText.text = " 刷新...";
        } else {
            pullDownRefresh.SetActive(false);
        }
    }
    void OnEndDrag() {
        if (findItemGridRoot.GetComponent<RectTransform>().anchoredPosition.y < 0f) {
            refreshText.text = " 刷新完";
        }
    }

// 某些属性，对抽象基类的抽象方法的实现与覆写
    public override string BundleName {
        get {
            return "ui/view/testview";
        }
    }
    public override string AssetName {
        get {
            return "TestView";
        }
    }
    public override string ViewName {
        get {
            return "TestView";
        }
    }
    public override string ViewModelTypeName {
        get {
            return typeof(TestViewModel).FullName;
        }
    }
    public TestViewModel ViewModel {
        get {
            return (TestViewModel)BindingContext;
        }
    }
    public override bool IsRoot {
        get {
            return true;
        }
    }
}
```

### 3.2 热更新视图的相关联的 ViewModel 实例一个：

```
public class GuideViewModel : ViewModelBase { // 把这些个不同类的继承关系串起来，怎么串起来呢？

    protected override void OnInitialize() {
        base.OnInitialize(); // <<<<<<<<<<<<<<<<<<<< 这里还是有点儿不懂
        Initialization();
        DelegateSubscribe();
    }
    void Initialization() {}
    void DelegateSubscribe() {}
}
```

### 3.3 static ViewManager：热更新里的视图管理类，持有所有需要热更新的视图的静态引用

```
using Framework.MVVM;
using Framework.Util;
// 它说，我是一个静态管理类，我要把每个需要热更新的视图都持有一个静态引用;
// 当需要实例化的时候，要么返回现持非空的视图，要么就新实例化一个该类型的视图
// 同样，与每个视图相绑定的是，MVVM 设计模式的 ViewModel，通过 UnityGuiView : IView<ViewModelBase> 抽象基类的继承实体类
// 热更新包里，看得是平淡无奇的 ViewModel，但是因为继承自 CrossBindingAdapter 的子类的 ViewModelAdapter，使用 View   怎么样呢？跨
// 面板管理器:  看上面，是完全可以用 framework 里定义的适配什么的呀
public static class ViewManager {
    // 这里固化适配为两维三维都可以
    public static Canvas UI2DRoot;
    public static Canvas UI3DRoot;
    public static RectTransform transfom;
    public static Dictionary<string, UnityGuiView> views = new Dictionary<string, UnityGuiView>();
    // public static TMP_FontAsset pingfangregularFont; // 把字体的部分都先简单地略过
    // public static TMP_FontAsset pingfangmediumFont;
    public static Font regularFont;
    public static Font mediumFont;

    public static void InitializeStartUI() {
        // LoadBaseAsset(); // 它说先加载场景以及视图里可能会用到的必要的字体，先。。。
        CreateBaseUI();
    }
    // static void LoadBaseAsset() {
    //     LoadFont(); // 这里说，就先加载一些字体
    // }
    // static void LoadFont() {  // 他们很喜欢苹果平方体
    //     pingfangregularFont = ResourceHelper.LoadTMP_FontAsset("ui/font/pingfangregular", "pic/ngfangregular", EAssetBun
    //     pingfangregularFont.material.shader = Shader.Find("TextMeshPro/Mobile/Distance Field");
    //     pingfangmediumFont = ResourceHelper.LoadTMP_FontAsset("ui/font/pingfangmedium", "pingfangmedium", EAssetBundleUn
    //     pingfangmediumFont.material.shader = Shader.Find("TextMeshPro/Mobile/Distance Field");
    //     regularFont = ResourceHelper.LoadFont("ui/font/regular", "regular", EAssetBundleUnloadLevel.Never);
    //     mediumFont = ResourceHelper.LoadFont("ui/font/medium", "medium", EAssetBundleUnloadLevel.Never);
    // }
// 仔细看这个方法：不是从热更新程序集里加载出 unity 里运行所需要的东西了吗？
    static void CreateBaseUI() {
        ResourceHelper.LoadCloneAsyn("ui/ui2droot", "UI2DRoot", // 明天上午去仔细追踪查看一下这个异步方法的细节
                                     (go) => {
            go.name = "UI2DRoot";
            GameObject.DontDestroyOnLoad(go);
            UI2DRoot = go.GetComponent<Canvas>();
            var viewRoot = new GameObject("ViewRoot"); // 实例化一个新空控件当作是视图层的根节点
            viewRoot.layer = LayerMask.NameToLayer("UI");
            var viewRect = viewRoot.AddComponent<RectTransform>();
            viewRect.SetParent(UI2DRoot.transform, false);
            viewRect.sizeDelta = new Vector2(0, 0);
            viewRect.anchorMin = Vector2.zero;
            viewRect.anchorMax = Vector2.one;
            viewRect.pivot = new Vector2(0.5f, 0.5f);
            poolRoot = new GameObject("PoolRoot").transform;
            poolRoot.SetParent(UI2DRoot.transform, false);
            poolRoot.gameObject.SetActive(false);
            ShowStartPanel();
        }, EAssetBundleUnloadLevel.Never);
    }
// 遍历当前视图管理器里所管理的所有的视图，凡是不是所指定特定视图的，一律隐藏起来（应该只是不让用户看见，它还在那里，在幕后的某个角落
    public static void CloseOtherRootViews(string viewName) {
        foreach (var view in views.Values)
            if (view.ViewName != viewName && view.IsRoot)
                view.Hide();
    }

// 这里应该是一个导航视图吧，猜测（不是视图，是 panel   ？）昨天晚上少眠，今天状态相对较差，期待明天会比较好
// 明天这些部分，今天所有有疑问的部分都再仔细地看一下
    static void ShowStartPanel() {
        GuideView.Reveal();
    }
#region Util
#endregion

#region GridItemPool
    public static Transform poolRoot;
// 这里所彩的数据结构栈：应该是与特定的应用特性相关的，能够保证后进先出和保证效率的
    public static Dictionary<string, Stack<GameObject>> gridItemPool = new Dictionary<string, Stack<GameObject>>();
    public static GameObject GetGridItemFromPool(string name) {
        if (gridItemPool.ContainsKey(name) && gridItemPool[name].Count > 0) {
            var gridItem = gridItemPool[name].Pop();
            return gridItem;
        }
        return null;
    }
    public static void CacheGridItemToPool(string name, GameObject go) {
        if (!gridItemPool.ContainsKey(name))
            gridItemPool[name] = new Stack<GameObject>();
        Stack<GameObject> goList = gridItemPool[name];
        go.transform.SetParent(poolRoot, false);
        goList.Push(go);
    }
#endregion

// 视图里的小物件管理，是热更新起始时重要的三个步骤之二：   可是仍然感觉他们只是很不起眼的一两个小物件，根本不值一提呀
// 这部分的细节暂时跳过，等改天实现自己游戏热更新需要参考的时候还可以修补上

#region Other
    static bool isOverUI = false;
    static bool isCheckedOverUI = false;
    static List<RaycastResult> raycastResults = new List<RaycastResult>();
    // 是否触摸到 UI 控件
    public static bool IsPointerOverUI() {
        if (isCheckedOverUI) {
            return isOverUI;
        }
        isCheckedOverUI = true;
        isOverUI = false;
        if (EventSystem.current.IsPointerOverGameObject()) {
            isOverUI = true;
        }
        PointerEventData pointer = new PointerEventData(EventSystem.current);
        pointer.position = Input.mousePosition;
        EventSystem.current.RaycastAll(pointer, raycastResults);
        if (raycastResults.Count > 0) {
            isOverUI = true;
        }
        TryStopTapEvent();
        return isOverUI;
    }
    public static void TryStopTapEvent() {
        CoroutineHelper.StartCoroutine(StopTapEvent());
    }
    static IEnumerator StopTapEvent() {
        yield return new WaitForEndOfFrame();
        isOverUI = false;
        isCheckedOverUI = false;
    }
```

```
#endregion

// 热更新的视图，远远不止这两个，但是留这两个已经够参考了，其它删除了
#region Views
    static TestView _testView;
    public static TestView TestView {
        get {
            if (_testView == null) {
                _testView = new TestView();
                _testView.BindingContext = new TestViewModel();
                views.Add(_testView.ViewName, _testView);
            }
            return _testView;
        }
    }
    static GuideView _guideView;
    public static GuideView GuideView {
        get {
            if (_guideView == null) {
                _guideView = new GuideView();
                _guideView.BindingContext = new GuideViewModel();
                views.Add(_guideView.ViewName, _guideView);
            }
            return _guideView;
        }
    }
#endregion
}
```

## 4 HotFixMain 静态类：热更新程序的入口

```
// 热更工程入口
public static class HotFixMain {
    public static void Start() {
        Debug.Log("InitializeTypeDatas");

// 反序列化场景专用资源包里的数据为 SceneTypeData 自定义类型，缓存在管理器宝典里，及时准备好以便实例化
        TypeDataManager.InitializeTypeDatas();
// 资源包里的小物件管理：反序列化为自定义的 TypeData 以便必要的时候可以第一时间顺利实例化（这部分逻辑被我跳过了，改天若是需要参考再
        ViewManager.InitializeItemDatas();
        Debug.Log("HotFixMain.Start()");
// 前面场景反序列化，小物件反序列化考都转为 TypeData，那么现在应该是可以初始化用户启动应用可以看见的第一屏屏幕了
        ViewManager.InitializeStartUI(); // 本质上也就是说，要热更新程序集里的视图管理类帮显示当前这个起始视图
    }
}
```

## 5 ILRuntime 类库里源码的基本理解，最重要的涉及到的相关的类与方法摘要

### 5.1 CrossBindingAdaptor : IType 跨域（程序集）绑定适配器 + CrossBindingAdaptorType interface 公用接口类（为什么要这个公用接口类？）

```
public interface CrossBindingAdaptorType {  // 公用接口类
    ILTypeInstance ILInstance { get; }
}

// This interface is used for inheritance and implementation of CLR Types or interfaces
public abstract class CrossBindingAdaptor : IType {
    IType type;

// 下面是定义的几个公用的抽象方法，供子类实现
    // This returns the CLR type to be inherited or CLR interface to be implemented
    public abstract Type BaseCLRType { get; }
    // If this Adaptor is capable to impelement multuple interfaces, use this Property, AND BaseCLRType should return null
    public virtual Type[] BaseCLRTypes {
        get {
            return null;
        }
    }
    public abstract Type AdaptorType { get; }
    public abstract object CreateCLRInstance(Enviorment.AppDomain appdomain, ILTypeInstance instance);

    internal IType RuntimeType { get { return type; } set { type = value; } }

// 反射机制的所有可能涉及的相关的方法定义; getters/setters
#region IType Members
    public IMethod GetMethod(string name, int paramCount, bool declaredOnly = false) {
        return type.GetMethod(name, paramCount, declaredOnly);
    }
    public IMethod GetMethod(string name, List<IType> param, IType[] genericArguments, IType returnType = null, bool declar
        return type.GetMethod(name, param, genericArguments, returnType, declaredOnly);
    }
    public List<IMethod> GetMethods() {
        return type.GetMethods();
    }
    public int GetFieldIndex(object token) {
        return type.GetFieldIndex(token);
    }
    public IMethod GetConstructor(List<IType> param) {
        return type.GetConstructor(param);
    }
    public bool CanAssignTo(IType type) {
        bool res = false;
        if (BaseType != null)
            res = BaseType.CanAssignTo(type);
        var interfaces = Implements;
        if (!res && interfaces != null) {
            for (int i = 0; i < interfaces.Length; i++) {
                var im = interfaces[i];
                res = im.CanAssignTo(type);
                if (res)
                    return true;
            }
        }
        return res;
    }
    public IType MakeGenericInstance(KeyValuePair<string, IType>[] genericArguments) {
        return type.MakeGenericInstance(genericArguments);
    }
    public IType MakeByRefType() {
        return type.MakeByRefType();
    }
    public IType MakeArrayType(int rank) {
        return type.MakeArrayType(rank);
    }
    public IType FindGenericArgument(string key) {
        return type.FindGenericArgument(key);
    }
    public IType ResolveGenericType(IType contextType) {
        return type.ResolveGenericType(contextType);
    }
    public IMethod GetVirtualMethod(IMethod method) {
        return type.GetVirtualMethod(method);
    }
    public void GetValueTypeSize(out int fieldCout, out int managedCount) {
        type.GetValueTypeSize(out fieldCout, out managedCount);
    }
// Getter / Setter s
    public bool IsGenericInstance {
        get {
            return type.IsGenericInstance;
        }
    }
    public KeyValuePair<string, IType>[] GenericArguments {
        get {
            return type.GenericArguments;
        }
    }
    public Type TypeForCLR {
        get {
            return type.TypeForCLR;
        }
    }
    public IType ByRefType {
        get {
            return type.ByRefType;
        }
    }
```

```
public IType ArrayType {
    get {
        return type.ArrayType;
    }
}
public string FullName {
    get {
        return type.FullName;
    }
}
public string Name {
    get {
        return type.Name;
    }
}
public bool IsValueType {
    get {
        return type.IsValueType;
    }
}
public bool IsPrimitive {
    get {
        return type.IsPrimitive;
    }
}
public bool IsEnum {
    get {
        return type.IsEnum;
    }
}
public bool IsDelegate {
    get {
        return type.IsDelegate;
    }
}
public AppDomain AppDomain {
    get {
        return type.AppDomain;
    }
}
public Type ReflectionType {
    get {
        return type.ReflectionType;
    }
}
public IType BaseType {
    get {
        return type.BaseType;
    }
}
public IType[] Implements {
    get {
        return type.Implements;
    }
}
public bool HasGenericParameter {
    get {
        return type.HasGenericParameter;
    }
}
public bool IsGenericParameter {
    get {
        return type.IsGenericParameter;
    }
}
public bool IsArray {
    get { return false; }
}
public bool IsByRef {
    get {
        return type.IsByRef;
    }
}
public bool IsInterface {
    get { return type.IsInterface; }
}
public IType ElementType {
    get {
        return type.ElementType;
        }
    }
    public int ArrayRank {
        get { return type.ArrayRank; }
    }
    public int TotalFieldCount {
        get {
            return type.TotalFieldCount;
        }
    }
    public StackObject DefaultObject {
        get {
            return default(StackObject);
        }
    }
    public int TypeIndex {
        get {
            return -1;
        }
    }
#endregion
}
```

## 5.2 委托适配器（DelegateAdapter）和委托转换器（DelegateConvertor）

- 如果只在热更新的 DLL 项目中使用的委托，是不需要任何额外操作的，就跟在通常的 C# 里那样使用即可
- 如果你需要将委托实例传给 ILRuntime 外部使用，那则根据情况，你需要额外添加适配器或者转换器。
- 如果在运行时发现缺少注册某个指定类型的委托适配器或者转换器时，ILRuntime 会抛出相应的异常，根据提示添加注册即可。
- 1. **委托适配器（DelegateAdapter）**
  - 如果将委托实例传出给 ILRuntime 外部使用，那就意味着需要将委托实例转换成真正的 CLR（C# 运行时）委托实例，这个过程需要动态创建 CLR 的委托实例。由于 IL2CPP 之类的 AOT 编译技术无法在运行时生成新的类型，所以在创建委托实例的时候 ILRuntime 选择了显式注册的方式，以保证问题不被隐藏到上线后才发现。委托适配器的具体用法会在下面的例子里面看到。
- 2. **委托转换器（DelegateConvertor）**
  - ILRuntime 内部是使用 Action, 以及 Func 这两个系统自带委托类型来生成的委托实例，所以如果你需要将一个不是 Action 或者 Func 类型的委托实例传到 ILRuntime 外部使用的话，除了委托适配器，还需要额外写一个转换器，将 Action 和 Func 转换成你真正需要的那个委托类型。委托转换器的具体用法会在下面的例子里面看到。

# 6 热更新模块总结

- 感觉源码相关的部分已经看得差不多了，感觉大部分懂了，
- 可是还是有不少部分似懂非懂，需要在自己小游戏的实现里自己写和实现的过程中再回来一再参考加深理解
- 明天会把这个热更新模块项目或是整个游戏框架疑难部分再多看一遍，
- 然后打算着写开始熟悉自己游戏的源码和进行改造
- 自己写和实现的过程中，应用会有狠多的知识点来考验自己
- 可是也只有自己真正写过一遍，才能真正理解得透彻呀。爱表哥，爱生活!!!

## 6.1