View Binding Model UI Framework

❖ 安装

从 Git 上下载工程，工程里的 Assets 下面有两个文件夹，一个是 Scripts，框架的代码，Samples 是框架的示例代码。把 Scripts 文件夹放到你的工程目录下，就可以使用了。

❖ UI 编辑

一、定义 Canvas 的层级结构

Canvas 的层级用来给 UI 分层，不同层的 UI 会有前后遮挡关系，比如示例项目里面的 ViewLayer，分为了背景层、前景层、常规层、弹出层、顶层和最顶层，背景和前景用来放一些美术要显示的效果图，比喻一个会移动的星空，月亮等，常规层用来放各个模块的 UI，比喻大厅的，角色信息，邮箱，商城等乖，弹出层用于显示弹出窗口，提示窗口，顶层最顶层可以用来弹 Tips 或者是紧急信息等，不同项目，会有不会的定义，这里只是抛砖引玉。ViewLayer 上面那个 Attribute 是用来在 ViewConfig 配置时，layer 这个选项会引用这个枚举内容。

```
[PropertyToEnumAttribute(typeof(ViewConfig), "layer")]
1 reference
public enum ViewLayer {
    0 references
    BackgroundLayer,
    0 references
    ForegroundLayer,
    0 references
    NormalLayer,
    0 references
    PopupLayer,
    0 references
    TopLayer,
    0 references
    MostTopLayer,
}
```

二、定义好 layer 之后，再来定义一下游戏的模块类型

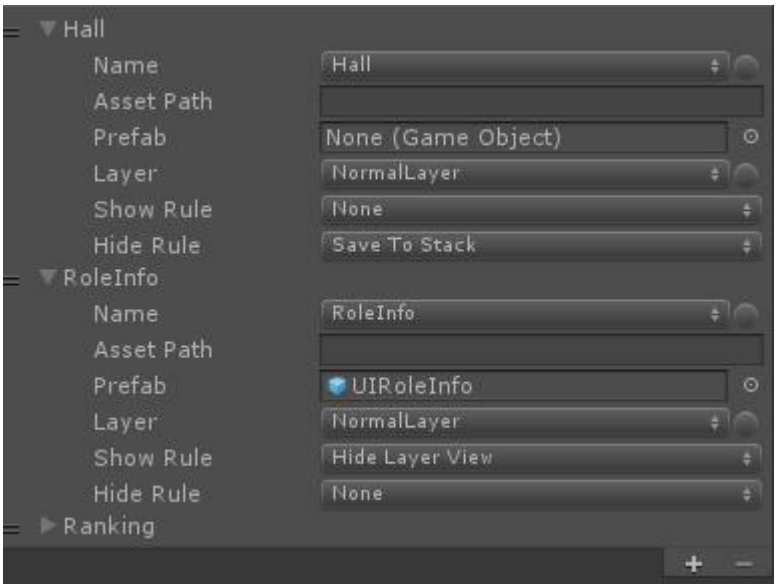
定义模块是为了方便配置和加载模块所用的 UI 资源和对应的 Model 以及 View 对象。参考示例：

```
[PropertyToEnumAttribute(typeof(ViewConfig), "name")]
4 references
public enum ViewModulesName {
    0 references
    Hall,
    2 references
    RoleInfo,
    0 references
    Mailbox,
    2 references
    Ranking,
    0 references
    Settings,
    0 references
    Friends,
}
```

游戏的模块可以分为大厅，角色信息，邮箱，排行榜，设置，好友等等。用法在后面创建 Model 和 View 时会有讲解，这个 PropertyToEnumAttribute 跟 ViewLayer 的 Attribute 一样，只是为了 ViewConfig 配置时，可以引用到自定义的枚举内容。

三、创建 ViewConfig

上面定义的 ViewLayer 跟模块名称都提到了这个东西，首先在 Unity 的 Project 里面使用右键菜单 Create/View Config Asset 来创建一个 ViewConfig。ViewConfig 的示例：



可以通过右下角的加号来添加一个 View 配置，减号删除一个 View 配置。各个参数的意思：

String Name　　模块名称，如果有定义模块枚举类型，右边将可以有下拉模块选择

String AssetPath　UI 的资源路径（目前还没实现该功能）

GameObject Prefab　　UI 的资源引用

Int Layer　　UI 对应的层级，如果有定义 ViewLayer 枚举类型，右边将有下拉层选择

ShowRule　　显示规则，就是当 UI 显示时，对应的行为

　　HideLayerView　　隐藏当前 Layer 的所有 View

HideRule　　隐藏规则，当 UI 隐藏时，对应的行为

　　SaveToStack　　保存到堆栈中，当上层带 HideLayerView 的 UI 隐藏时，自动从堆栈中弹出，重新显示出来

ViewConfigAsset 可以有多份，比喻大厅的所有 UI 是一份，战斗里面的 UI 是一份，当场景切换后，把不需要的那一份对应的资源卸载掉。

四、创建一个 Model

从这一步开始，已经是开始动工作项目了，首先，所有 Model 都必须继承自 VBM.Model 这个类，这是一个抽像类，一般不使用这个什么功能都没有类，而是继承他的子类 DictionaryModel，这是一个字典类，提供 GetProperty 和 SetProperty 功能，例如我们要定义一个 RoleInfoModel 类：

```
public class RoleInfoModel : DictionaryModel {
    2 references
    public Sprite headIcon {
        get { return GetProperty<Sprite>("headIcon"); }
        set { SetProperty("headIcon", value); }
    }

    2 references
    public string username {
        get { return GetProperty<string>("username"); }
        set { SetProperty("username", value); }
    }

    2 references
    public int goldCount {
        get { return GetProperty<int>("goldCount"); }
        set { SetProperty("goldCount", value); }
    }
```

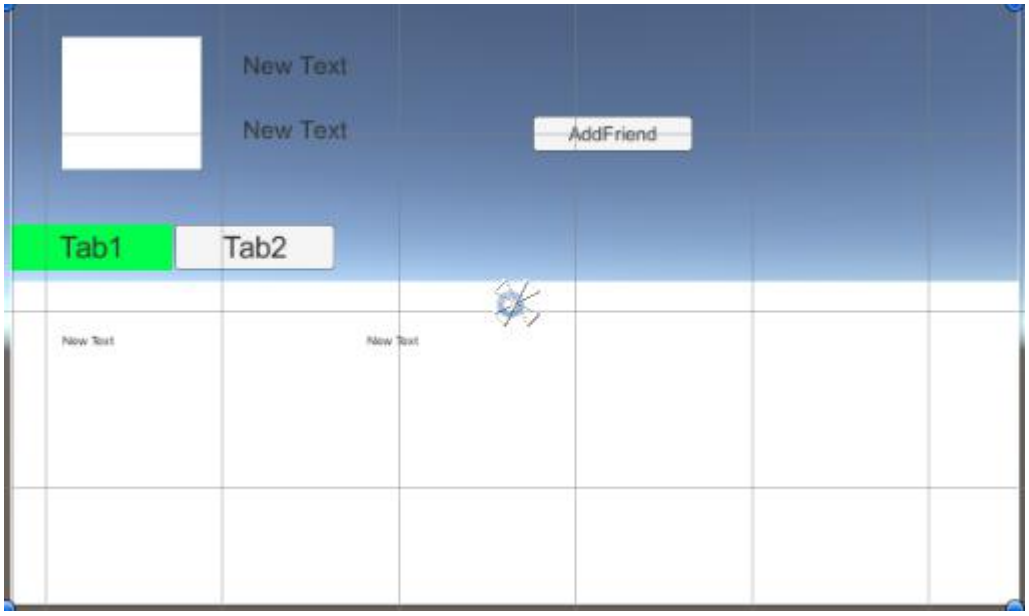
上面有三个属性，分别是 headIcon, username, goldCount，分别调用父类的 GetProperty 和 SetProperty 函数，所起来是不是很不爽，没办法，C#不支持 Property 自动 NotifyChange，所以只好包了一层，当然外部使用时还是正常的，只是定义时，会让有强迫症的家伙想毁灭掉这玩艺儿的心都有。（说点题外话，C#这么定义会很丑，但如果用 lua 来实现，却又是那么自然）

当然了，一个模块不可能只有属性，没有行为，比喻角色 UI 上就有一个添加好友的按钮，可以让你添加当前看的角色信息为好友，所以还需要再定义一个函数来应用 UI 的添加好友按钮的事件。

```
    0 references
    public void AddFriend() {
        Debug.Log("Add friend function.");
    }
```

五、创建一个 ViewModelBinding

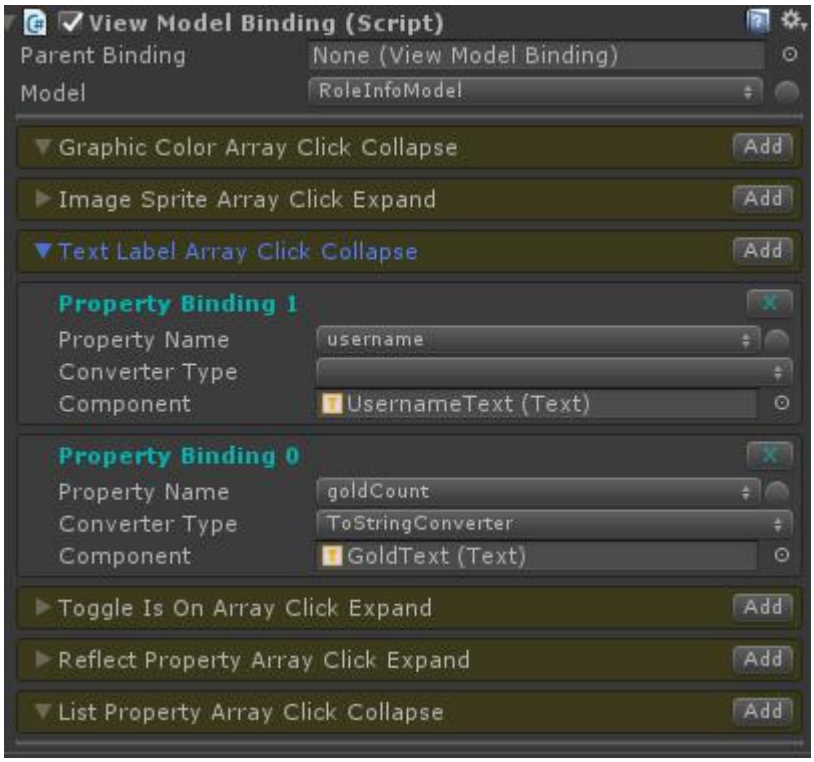
OK，终于讲到 UI 了，首先让我们来拼一个角色信息的 UI：



左上角呢，表示角色头像，然后是用户名，金币数，右边是添加好友按钮，下面两个 Tab 是两个标签页，内容是什么不重要，重要的是演示的功能是怎么样。

有了这个 UI 以后，就要把这个 UI 绑定到 Model 上，首先要在 UI 上 Add Component，在弹出来的菜单里面搜索 ViewModelBinding，回车后就创建了一个 ViewModelBinding 了。我们来看一下 ViewModelBinding 的样子，首先不要问我为什么会显示这么多种类型，他们看起来都来自于同一个父类，为什么不做成一种，我也想做成一个右键菜单，然后添加一个 PropertyBinding，然后根据类型不同，自动分类，只是 Unity 不支持多态序列化，我也是迫不得以而为之。抱歉，跑题了。

让我们看一下 ViewModelBinding 支持的类型，目前是 GraphicColor、ImageSprite、TextLabel、ToggleIsOn，ReflectProperty，ListProperty 这几种，未来肯定会扩展，只是大部分情况下只需要用到 TextLabel 和 ImageSprite。说一下各类型的意思吧。



GraphicColor	就是绑定一个 Graphic 对象的 Color，Image，Text 等控件都是来自于 Graphic 对象的。
ImageSprite	就是绑定一个 Image 对象的 Sprite
TextLabel	绑定一个 Text 的 text，因为是 Text 对象的 text 属性，TextText 叫起来很奇怪，所以就干脆叫 TextLabel
ToggleIsOn	绑定 Toggle 对象的 IsOn 属性
ReflectProperty	绑定一个 Componet 对象的一个属性，只要这个属性支持反射。
ListPropety	绑定一个 List，这个后面在讲排行榜示例的时候，会详细说明。

绑定的参数一般有 3 个：

PropertyName	就是 model 的属性，刚刚定义那种让强迫症想发疯的 Property 属性，注意 Field 属性是不能被绑定的
ConverterType	转换类型，一个 Model 数据，一个 View 数据，总会有那么一些是匹不上类型的，比喻金币要显示到 UI，就要把 int 转成 string，所以这个时候就要用到这个转换类型了。目前的转换类型有：

ToStringConverter	就是一个对象的 ToString
Float1StringConverter	格式化 Float，保留一位小数位，然后转成 String
Float2StringConverter	格式化 Float，保留两位小数位，然后转成 String
Float2StringConverter	格式化 Float，保留三位小数位，然后转成 String
FloorStringConverter	向下舍入转成 String
CeilStringConverter	向上舍入转成 String

当然这些转换不能满足各自的需求，所以有需求的，自己继于 Property Converter 类，照着上面随便一个类画一个就行了。

注意：以后的所有绑定，都是由 Model 数据来更新 UI 绑定对应的属性，UI 属性如果被修改，是不会影响到 Model 的数据的

ViewModelBinding 还有最最重要的两个属性:

ParentBinding	就是父亲 ViewModelBinding，用来作延迟加载等相关功能，下面马上讲到
Model	当你定义了一个 Model 以后，自然可以在编辑器看到定义的 Model，选择要绑定的 Model，如果这个 UI 是要绑定多个 Model，那就创建多个 ViewModelBinding，一个 ViewModelBinding 只能绑定一个 Model，而也只有绑定了 Model，才能绑定属性。

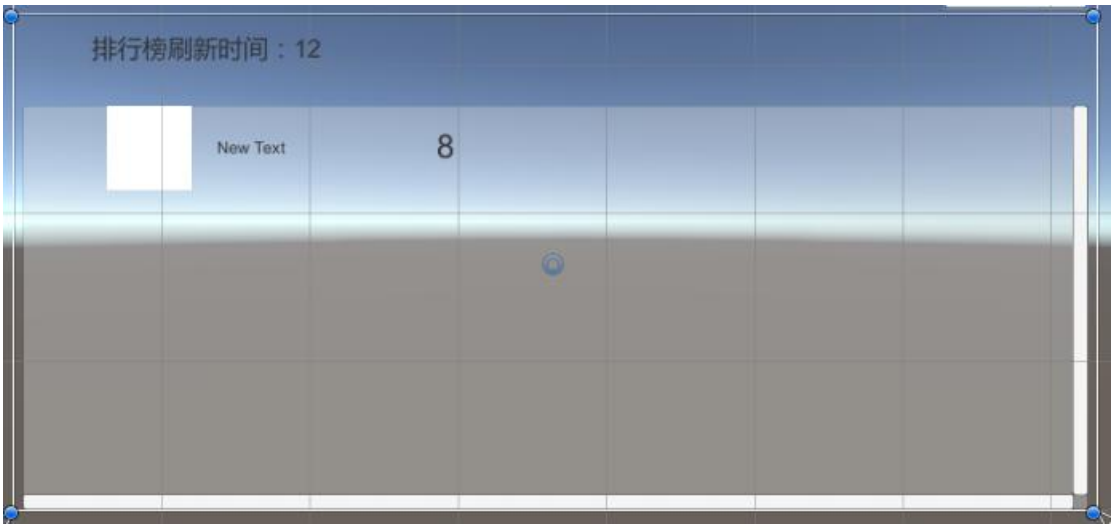
六、延迟加载

延迟加载就是一个 UI 可以只要需要显示的时候才被创建，然后自动根据绑定属性来刷新 UI，或者是 UI 内的某一个标签页只有当需要的时候，才会刷新数据。还是以上面那个角色信息的 UI 为示例，角色信息 UI 有两个标签页内容，Tab1 和 Tab2，对应了 Content1 和 Content2，看一下 UIRoleInfo 的 Hierarchy 结构：

默认会显示每一个 Tab1 对应的内容 Content1，但 Content2 只会在用户点击显示 Tabl2 后才会被初始化，UI 数据才会被刷新，那这个延迟加载是怎么做的呢，很简单，在 Content1 和 Content2 上面分别挂一个 ViewModelBinding，根据 Unity 对于 Component 的规则，只有当一个 Component 被显示时，才会激活它，不显示，永远不会激活。

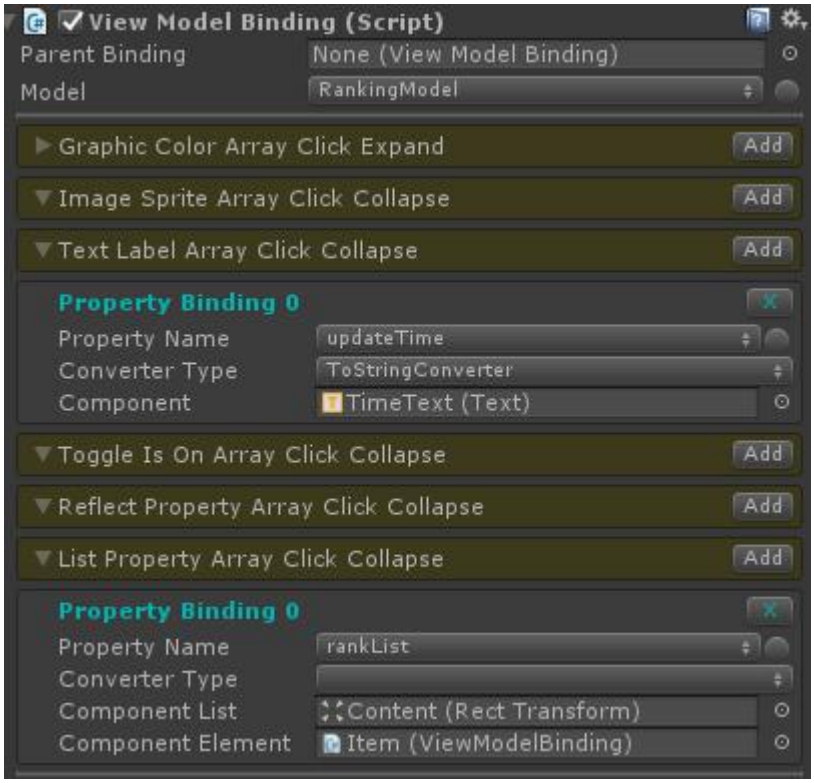
七、列表的支持

列表就是一个 List，对于类似于排行榜这种需要有多条内容显示的支持，因为 List 要有各种 Add，Remove，Insert 等操作增加事件回调，所以不用使用原生的 List，只能使用 ListModel，而且每一个元素都必须是一个 Model，因为每个元素的内容都是单独更新，当然不用担心 Model 问题，因为这是一个非常非常轻量级的对象，随便继承，随便使用。下面以排行榜 UIRaning 为例，讲一下 List 怎么使用：



首先看一下这个排行榜，（很抱歉，没有找美术资源，就这么丑，但美观问题不影响实际功能，大家将就着看。），有一个刷新时间，还有一个 ScrollView，里面的元素是一个头像和一个用户名，再加一个排名。

再看一下这个排行榜的 ViewBindingModel 的情况：



一个 TextLabel 的属性来绑定排行榜的更新时间，最下面就是 List 绑定了，rankList 是 RankingModel 的一个属性，ComponentList 就是 UIRanking ScrollView 里面的 Content，CompnentElemtn 使用的是 Conent 的 Item。

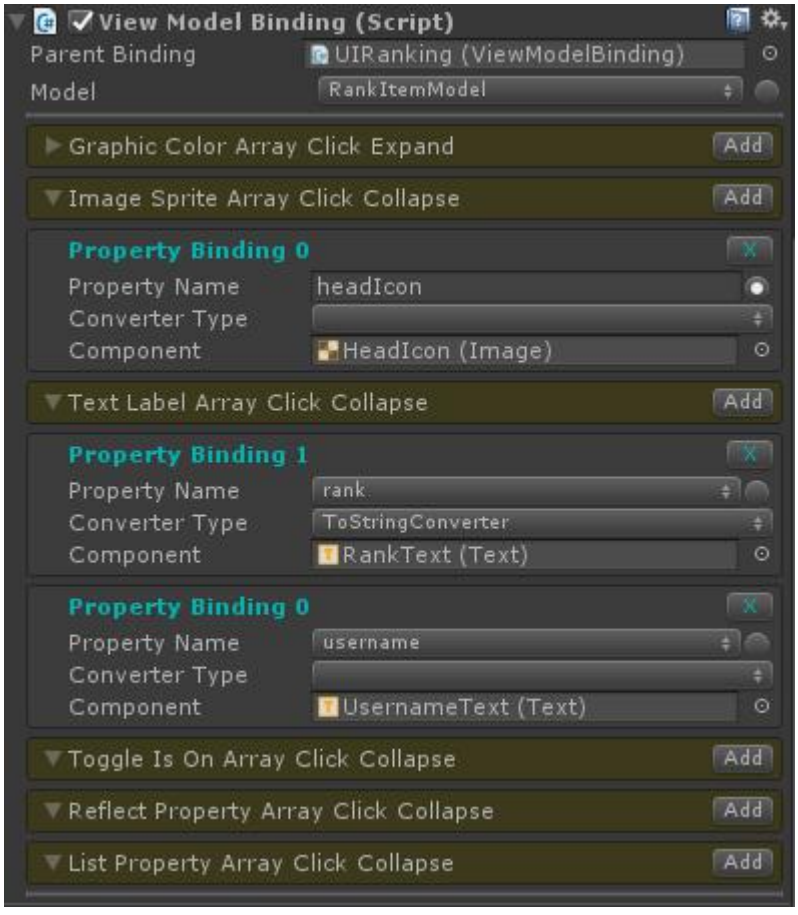
```
public class RankingModel : DictionaryModel {
    2 references
    public int updateTime {
        get { return GetProperty<int>("updateTime"); }
        set { SetProperty("updateTime", value); }
    }

    5 references
    public ListModel rankList {
        get { return GetProperty<ListModel>("rankList"); }
        set { SetProperty("rankList", value); }
    }
}
```

- 这是 RankingModel 的代码定义，一个是更新时间 UpdateTime，一个是 ListModel 类型的 rankList。

- 这是 UIranking 的 Hierarchy 结构

到这里，ListModel 就绑定完了，其实就是定义一个 ListModel 属性，然后在 UI 上使用 ListProperty 把这个属性绑上去，然后选择 ListModel 对应的 Transform，和元素对应的 Transform 就完事了。还有一点，就是 ListModel 的元素都是 Model 类型，所以对应的 UI 上也需要绑定一个 ViewModelBinding。我们来看一下这个 UIRanking 的 Content 下面的 Item 相应的 ViewModelBinding 内容：



看懂了上面的角色绑定以后，这里的绑定就跟上面的一个模样，对于两个 TextLabl 和一个 ImageSprite 进行绑定，到处 ListModel 的绑定完整结束。用户不需要关心这个 UI 的事情了，只要 ListModel 有 Add，Remove，Sort 等操作，UI 都会有对应的显示。

❖ 代码编写

一、初始化

首先呢，需要初始化 ViewManager 的 Canvas 的 Layer 层次，然后就是初始化 ViewConfig 了

```
ViewManager.Instance.InitCanvasLayers(canvas, typeof(ViewLayer));
ViewConfigAsset configAsset =
    Resources.Load<ViewConfigAsset>("Configs/ViewConfigAsset");
ViewConfigManager.Instance.AddConfigs(configAsset.configs);
```

二、创建 Model

```
ModelManager.Instance.CreateModel<RoleInfoModel>();
ModelManager.Instance.CreateModel<RankingModel>();
```

可能是通用 GetModel 函数再次获得 model，注意 Model 使用的是 string 来做唯一 Id 的，不能有重复。

```
RoleInfoModel model = ModelManager.Instance.GetModel<RoleInfoModel>();
```

三、创建 View

```
ViewConfig roleInfoConfig =
    ViewConfigManager.Instance.GetViewConfig(ViewModulesName.RoleInfo.
ToString());
ViewConfig rankingConfig =
    ViewConfigManager.Instance.GetViewConfig(ViewModulesName.Ranking.
ToString());
ViewManager.Instance.CreateView<TransientView>(roleInfoConfig);
ViewManager.Instance.CreateView<TransientView>(rankingConfig);
```

创建 View 需要一个 ViewConfig 对象，ViewConfig 对象就是外部配置的。CreateView 接口可以创建 View 以及 View 的子类，这里的 `TransientView` 是一个特殊的 View，会自动加载和支持自动卸载 UI 资源的功能。

❖ 示例演示

文档里所有的示例都是来自于 Samples 里的，可以打开场景，可看源码获得更多功能与支持。感谢收看，谢谢。