客户端框架

1、简介

框架内容包括：资源加载、缓存以及对象池，资源热更。UI 界面加载、销毁管理。UI 节点使用容器查找。重写 Button、Toggle、Selectable 组件实现更为广泛的功能需求。Socket 网络通信。Protobuf 网络传输。屏幕自适应。加载 Excel 表格。一些工具组件，如 Text 框颜色渐变、物体自传、帧动画组件等。日志管理器。文件读取。声音管理。按钮事件管理。本地存储管理。按键点击事件管理。无限循环列表。全局的提示窗口、网络加载窗口、Loading 窗口。Dotween 动画管理。

2、资源管理

负责资源管理的组件为“ObjectCache”，“ObjectCache”组件为单例组件，所以可以直接调用公共成员。资源管理分为资源加载、缓存和对象池。

2.1、资源加载

资源加载分为 Resources 加载和 Assetbundle 加载。

2.1.1、Resources 加载

需要将资源提前放置于 Resources 文件夹下，随后才能进行动态加载。

2.1.1.1、LoadResource 同步加载资源

```
public T LoadResource<T>(string assetName) where T : Object
```

assetName：加载的资源名字

2.1.2、Assetbundle 加载

此加载方式适用于热更资源加载。关于 Assetbundle 加载和热更将在另一个文档解释。

2.2、资源缓存

资源缓存是在界面切换之前，把资源从硬盘缓存到内存中，从而在游戏界面加载物体时，节省加载时间。资源缓存需要重写 UIManager 的 CacheAssets 属性，传入需要缓存的所有资源名字。返回界面的时候，自动销毁上一个界面的缓存资源。

2.1.3、对象池

对象池用于需要频繁创建和销毁的物体，此处设计只能存储 GameObject。对象池传入名字后是去缓存里查找对应资源，所以对象池的资源需要预先缓存。对象池需要先创建，否则直接取用返回 null 并警告提示。对象池创建的时候是有大小的，每获取一个对象，List 里就移除掉一个对象，所以，当取用完之后，需要重新添加对象到对象池，或者回收对象，或者取用对象的时候用参数控制是否添加新对象到对象池。最后，释放对象池，可以手动释放，界面返回的时候，也会自动释放掉上一个界面的对象池。

2.1.3.1、CreateGameObjectPool 创建对象池

所有对象使用 List 存储，在创建的时候有初始大小。已有的对象池不会再次创建。

```
public void CreateGameObjectPool(string assetName, int count)
```

assetName：资源名字。如果已经存在“assetName”对象池，则警告提示

count：对象池大小

2.1.3.2、CheckIsExistGameObjectPool 是否存在对象池

```
public bool CheckIsExistGameObjectPool(string assetName)
```

assetName：资源名字。

count：对象池大小

2.1.3.3、CheckIsGameObjectPoolFullUse 对象是否使取完

```
public bool CheckIsGameObjectPoolFullUse(string assetName)
```

assetName：资源名字。

2.1.3.4、AddGameObjectToPool 添加物体到已有对象池

```
public void AddGameObjectToPool(string assetName, int count = 1)
```

assetName：资源名字。

count：添加的个数

2.1.3.5、GetGameObjectFromPool 从对象池获取对象

```
public void GetGameObjectFromPool(string assetName, bool active = true, bool autoAdd = true)
```

assetName：资源名字。

active ：物体加载出来是否可见

autoAdd ：对象池大小不够时是否自动添加新对象到对象池

2.1.3.6、Recycle 回收到对象池

```
public void Recycle(GameObject obj)
```

obj：回收的对象。根据对象名字回收物体。

3、界面加载

界面存储结构为栈，加载界面进栈，返回界面出栈。UIController 负责界面加载、返回。所有窗口命名规则:“ui_win_”+窗口名字，所有窗口都需要挂载一个继承自 UIManager 的组件，这个组件命名规则：窗口名字+“UIManager”。UIController 会在依次调用到 UIManager 里的 UI 加载、数据加载等虚方法。界面加载分为两类，弹出界面和跳转界面。

3.1、弹出界面

弹出界面适用于在加载新窗口，而原来的窗口依然可见。

```
public void ShowWindow(string targetWindowName, string fromWindow = "", object[] args = null,
Action onFinished = null, bool isShowLoading = false, bool isDestroy = false, bool
isHideFormer = false)
```

targetWindowName：跳转的窗口名字，名字以“ui_win_”开头

fromWindow ：调用跳转窗口名字

args ：传递的参数

onFinished ：跳转成功的回调

isShowLoading ：是否显示 Loading 界面

isDestroy ：窗口加载完后，上一个窗口是否被销毁

isHideFormer ：是否隐藏上一个界面

3.2、切换界面

切换界面适用于在加载新窗口，而原来的窗口被隐藏掉。

```
public void ChangeWindow(string targetWindowName, string fromWindow = "", object[] args =
null, Action onFinished = null, bool isShowLoading = true, bool isDestroy = false, bool
isHideFormer = true)
```

参数与 ShowWindow 一样，只是缺省值不同。

3.3、返回界面

```
public void ReturnBack(int depth, Action onFinished, string fromWindow = "")
```

depth：返回 depth 个界面

onFinished：返回成功回调

fromWindow ：调用返回界面

```
public void ReturnBack(string targetWindow, Action onFinished, string fromWindow = "")
```

targetWindow：返回到指定名字界面

3.4、UIManager 里的重要方法

UIManager 提供了公共虚方法，供 UIController 调用，实现 UIController 对 UIManager 的管理，同时，继承 UIManager 的组件重写虚方法，达到加载 UI 和数据的目的。

3.4.1、缓存资源列表

子类重写此属性即可实现资源的缓存

```
 public virtual string[] CacheAssets
    {
        get { return null; }
    }
```

3.4.2、加载 UI

执行顺序第一

```
public virtual void InitUI()
```

3.4.2、加载数据

执行顺序第二

```
public virtual void InitData(object[] args)
```

args：跳转窗口的传入数据

3.4.3、OnEnable

执行顺序第三，此方法为 Unity 内置方法

```
protected virtual void OnEnable()
```

3.4.3、返回当前界面

当前界面再次成为主界面的时候调用

```
public virtual void ReturnBackToThisWindow(bool isChangeWindow)
```

isChangeWindow：是否是切换界面

3.5、切换焦点

UIController 里有一个属性：CurrentSelectedObj。此属性定位 unity 当前所选焦点。

3.6、重写组件

与焦点密切相关的就是可交互组件，重写的可交互组件有 Button、Toggle、Selectable 分别对应 MyButton、MyToggle、MySelectable。这些组件重写的目的是为了实现可交互组件在被选中时，触发相应的操作，如显示选中框，显示选中声音，给事件管理器传递弃选组件和被选组件等。

3.7、屏幕自适应

此处使用九宫格作为自适应机制，需要适应位置的物体需要挂载“ScreenAdapter”组件，然后选择适应位置。

```
public enum E_Pos
{
    Top_Left,//上左
    Top_Center,//上中
    Top_Right,//上右
    Center_Left,//中左
    Center_Center,//中
    Center_Right,//中右
    Bottom_Left,//下左
    Bottom_Center,//下中
    Bottom_Right,//下右
}
```

3.8、寻找 UI 节点

Button、Text、Image 等物体对应不同容器，这些容器挂载到窗口上，然后将窗口上的节点拖拽到该容器里，寻找 UI 节点即可根据容器的里每个节点所在 Index 查找到对应节点，且当编辑器里节点名字修改也不影响节点的查找。

4、常驻窗口

常驻窗口就是在每个界面都有可能出现的窗口，挂载的是一个继承自 SingletonWindow 的单例组件。常驻窗口包括：通用提示框、资源加载界面、网络加载界面、跑马灯。SingletonWindow 提供了“Show”显示界面的虚方法，以及隐藏界面的虚方法，具体每个界面根据自身功能增加新的成员。

5、网络通信

网络通信提供了 Http 通信和 Socket 通信。

5.1、Http 通信

Http 通信使用 Protobuff 实现短连接通信，负责管理 Http 通信的组件是“HttpWrapper”，这是一个单例组件，里面有一个公共方法实现消息请求：

```
public void HttpRequest<TReq, TResp>(string url, TReq req, Action<TResp> onFinished)
```

url：协议 Id

req：请求内容

onFinished：请求完成的回调

5.1、Socket 通信

Socket 通信使用 Protobuff 实现长连接通信，负责管理 Socket 通信的组件是“SocketWapper”，这是一个抽象的单例组件，需要有子类继承使用。消息内容为：协议 Id（1 个字节，大端）+消息长度（1 个字节）+消息内容。Socket 通信分为两个模块，消息发送和消息接收。

5.1.1、消息发送

两个重载公共方法实现消息请求：

```
public virtual void DoSocketRequest<TReq>(int protoId, TReq content, params int[] responseProtoIds) where TReq : class
```

protoId：协议 Id

content：请求内容

responseProtoIds：请求完成后会返回的消息 Id 列表

此请求会显示网络加载界面。同一条请求可能会返回不同的消息，所以使用“responseProtoIds”识别消息的返回，达到消息请求时显示网络加载界面，消息回复时关闭网络加载界面。

```
public virtual void DoSocketRequest<TReq>(int protoId, TReq content) where TReq : class
```

此请求不会显示网络加载界面。

5.1.1、消息接收

消息接收需要在继承“SocketWapper”组件的字类里添加自定义事件，接收到对应消息后，会自动调用对应协议 Id 的事件。例如被踢下线消息：

```
/// <summary>
/// 操作异常，被踢下线
/// </summary>
[ResponseEvent(ProtoId.KICK_OFF)]
public event Action<KickProto> HandleKickOff;
```

所有消息接收均为事件，且必须有特性“ResponseEvent”作为标识，“rotoId.KICK_OFF”为被踢下线的协议 Id。当接收到协议 Id 为“ProtoId.KICK_OFF”的消息时，“HandleKickOff”事件里注册的所有方法会被调用。

6、加载 Excel 表格

所有表格放置于“Assets/Doc/ResTable/”文件夹下。表格模块分为生成 Excel 表格为.res 文件和加载表格数据。

6.1、生成.res 文件

编辑器选择“MyTools/Convert Resource excel(.xlsx) to pb”，在“Assets\StreamingAssets\LocalResource.bundle\ResTable”下生成“resRoot.res”文件。

6.1、加载表格数据

ResTableContainer 类负责加载表格数据。

```
public static IResTable GetTable(string tableName)
```

tableName：表名

IResTable 为加载的表格数据。

7、工具类

工具类放置于“Assets\Scripts\Framework\Utils”文件夹下，有物体自传、自动加载帧动画图片等。

8、日志管理

所有的日志输出都调用“LogUtil”类，LogUtil 是一个静态类。LogUtil 提供了日志输出方法 Log、LogWarning、LogError、LogException。日志等级可以配置，配置文件路径为：Assets\StreamingAssets\LocalResource.bundle\config\config.json，LogLevel 等级：

```
public enum E_LogLevel : byte
    {
        None = 0,//不输出任何日志
        Exception = 1,//输出异常日志
        Error = 2,//输出错误日志
        Warning = 3,//输出警告日志
        Info = 4,//输出所有日志
        }
```

9、文件加载

9、10、11、12 将在后续更新。

9、本地内存管理

10、按键事件管理

11、DoTween 动画管理

12、声音管理