GameFrameworkEntry（框架入口）

GameFrameworkEntry使用链表来维护所有的GameFrameworkModule（框架模块），并提供对GameFrameworkModule的各种相关操作以及框架版本号信息

1.只有在获取模块时对应模块不存在才会创建模块

2.创建模块时，根据模块优先级决定新模块在链表中的位置

3.轮询模块时，根据优先级决定轮询顺序（即正向遍历链表调用OnUpdate）（**在UGF中，由BaseComponent来调用轮询方法**）

4.**所有的Manager类都需要继承GameFrameworkModule**，并实现对应的IManager接口

GameEntry（游戏入口）

示例程序Start Force里**包含两个GameEntry**，一个是UGF的，一个是Start Force的。Start Force在场景GameFramework中有一个Game Framework物体，上面挂载了Start Force的GameEntry脚本作为游戏入口

1.UGF的GameEntry使用链表来维护所有的GameFrameworkComponent（框架组件），并提供各种相关操作。

2.所有GameFrameworkComponent在Awake中调用UGF的GameEntry的RegisterComponent(GameFrameworkComponen)，将自身添加到UGF的GameEntry中的链表里

3.Start Force的GameEntry脚本持有所有GameFrameworkComponent的引用，被分成了3个部分，分别负责调用Start方法，初始化内置GameFrameworkComponent，初始化自定义GameFrameworkComponent

4.GameFramework物体下有预制体Builtin（由UGF框架提供，原名GameFramework），该预制体下的所有子物体分别挂载了所有的内置GameFrameworkComponent。**所有内置GameFrameworkComponent都持有一个对应IManager接口的引用（可以视作GF的Manager在UGF中的代理或实现）**，在初始化时通过GameFrameworkEntry来获取实例

5.**自定义的GameFrameworkComponent**需要另外创建一个空物体作为GameFramework的子物体，然后将自定义组件各自挂载到空物体下，作为其子物体

FSM（有限状态机）与Procedure（流程）

流程是贯穿游戏运行时整个生命周期的有限状态机。框架用流程来处理所有的事情，不同的流程负责不同的工作，流程的切换是用有限状态机来实现的

1.FSM模块主要由状态，状态机，状态机管理器三部分组成

2.状态FsmState<T>维护一个事件码与事件处理方法的字典，并提供各种对应操作

3.状态机Fsm继承FsmBase并实现IFsm<T>接口，维护该状态机的所有状态与状态机数据，并提供各种对应操作

4.状态机管理器FsmManager维护所有状态机，并提供各种对应操作

5.Procedure模块主要由流程基类和流程管理器两部分组成

6.流程基类ProcedureBase继承自FsmState<IProcedureManager>，可以理解为一种特殊的状态

7.流程管理器ProcedureManager持有FsmManager与自身的Fsm的引用，并提供对流程的相关操作

8.**在UGF中的ProcedureComponent，主要负责读取创建好的流程类**，然后调用ProcedureManager来创建并开始状态机

DataNode（数据结点）

DataNode将任意类型的数据以树状结构的形式进行保存，用于管理游戏运行时的各种数据

数据结点的使用非常灵活，有以下三种使用方式：

使用数据结点组件，直接通过绝对路径获取或设置数据；

使用数据结点组件，通过参照某个数据结点和相对路径获取或设置数据；

使用数据结点组件获取数据结点后，通过数据结点的接口进行更多操作。

1.DataNode模块由数据结点和数据结点管理器两部分组成

2.数据结点（DataNode）存储数据以及父结点、子结点的相关信息

3.数据结点管理器（DataNodeManager）管理根结点，并提供数据结点的相关操作

ReferencePool（引用池）

ReferencePool功能负责管理对象的引用

1.ReferencePool功能主要由引用接口，引用集合，引用池三部分组成

2.只有实现了引用接口（IReference）的类才能被ReferencePool管理，在框架中由BaseEventArgs实现，也就是说**ReferencePool主要拿来管理对BaseEventArgs的引用**

3.引用集合（ReferenceCollection）维护一个IReference的队列，并提供该对象的各种信息或相关操作

4.引用池（ReferencePool）维护一个ReferenceCollection的字典，并提供各种相关的操作

Event（事件）

Event是游戏逻辑监听、抛出事件的机制。

1.Event模块由事件，事件池，事件管理器三部分组成

2.事件基类BaseEventArgs继承GameFrameworkEventArgs（还有另一种事件类也继承GameFrameworkEventArgs，**那种事件不受Event模块管理，是模块局部事件**）

3.事件池EventPool<T>主要维护事件结点（Event）（Event是对BaseEventArgs的封装）的队列（该队列处理线程安全的事件抛出）与一个事件码与事件处理方法的字典，并提供各种对应操作

4.事件管理器EventManager是对EventPool<T>里各种操作的代理

ObjectPool（对象池）

ObjectPool提供对象缓存池的功能，避免频繁地创建和销毁各种游戏对象，提高游戏性能。

1.ObjectPool模块主要由对象基类，内部对象，对象池，对象池管理器三部分组成

2.对象基类（ObjectBase）是所有需要由ObjectPool模块管理的对象的父类

3.内部对象（Object<T>）存储ObjectBase相关数据，并提供获取与回收的方法

4.对象池（ObjectPool<T>）使用链表维护池子里的所有Object<T>，并提供各种相关操作

5.对象池管理器（ObjectPoolManager）使用字典维护所有ObjectPool<T>，并提供各种相关操作

Task（任务）

Task功能主要负责管理Web请求、资源的下载或加载的任务的执行

1.Task功能主要由任务，任务代理，任务池三部分组成

2.任务（ITask）存储了任务执行需要的数据

3.任务代理（ITaskAgent<T>）是对ITask的代理类，提供对任务的各种对应操作

1. 任务池TaskPool<T>负责维护三种容器（可用任务代理，工作中任务代理，等待任务），并提供各种对应操作

Download（下载）

Download模块提供下载文件的功能，支持断点续传，并可指定允许几个下载器进行同时下载。更新资源时会主动调用此模块

1.DownLoad模块主要由下载任务，下载代理辅助器，下载任务代理，下载管理器，下载事件五个部分组成

2.下载任务（DownloadTask）实现了ITask接口，保存了下载任务的相关数据

3.下载代理辅助器（IDownloadAgentHelper）负责进行**实际的下载**逻辑处理（**在UGF中提供了默认的实现DefaultDownloadAgentHelper，使用WWW类进行下载**）（**下载过程中通过3个委托向DownloadAgent通知下载事件**）

4.下载任务代理（DownloadAgent）实现了ITaskAgent接口，负责处理下载任务，并持有一个IDownloadAgentHelper，调用其中的方法进行实际的下载。（**下载过程中通过4个委托向DownloadManager通知下载事件**）

5.下载管理器（DownloadManager）维护一个DownloadTask的TaskPool，对外提供DownloadTask的相关操作（**下载过程中通过4个委托将DownloadAgent的下载事件通知给UGF的DownloadComponent，然后由其派发4个全局事件**）

6.下载事件分为三部分，一部分是DownloadManager里的事件，另一部分是DownloadAgentHelper里的事件，这两部分定义在GF里，是**模块局部事件**，第三部分则是**UGF里定义的全局事件**,由DownloadComponent接收到模块局部事件后进行派发

WebRequest（Web请求）

WebRequest模块提供使用短连接的功能，可以用 Get 或者 Post 方法向服务器发送请求并获取响应数据，可指定允许几个 Web 请求器进行同时请求

1.WebRequest模块主要由由Web请求任务，Web请求代理辅助器，Web请求任务代理，Web请求管理器，Web请求事件五个部分组成

2.Web请求任务（WebRequestTask）实现了ITask接口，保存了Web请求任务的相关数据

3.Web请求辅助器（IWebRequestAgentHelper）负责进行**实际的请求**逻辑处理（**在UGF中提供了默认的实现DefaultWebRequestAgentHelper，使用WWW类进行请求**）（**请求过程中通过2个委托向WebRequestAgent通知请求事件**）

4.Web请求任务代理（WebRequestAgent）实现了ITaskAgent接口，负责处理请求任务，并持有一个IWebRequestAgentHelper，调用其中的方法进行实际的请求。（**请求过程中通过3个委托向WebRequestManager通知请求事件**）

5.Web请求管理器（WebRequestManager）维护一个WebRequestTask的TaskPool，对外提供WebRequestTask的相关操作（**请求过程中通过3个委托将WebRequestAgent的请求事件通知给UGF的WebRequestComponent，然后由其派发3个全局事件**）

6.下载事件分为三部分，一部分是WebRequestManager里的事件，另一部分是WebRequestAgentHelper里的事件，这两部分定义在GF里，是**模块局部事件**，第三部分则是**UGF里定义的全局事件**,由WebRequestComponent接收到模块局部事件后进行派发

Resource（资源）

DataTable（数据表）

DataTabel模块可以将游戏数据以表格（如 Microsoft Excel）的形式进行配置后使用。数据表的格式是可以自定义的。

1.DataTabel模块主要由数据行，数据表，数据表辅助器，数据表管理器，加载数据表事件五个部分组成

2.数据行（IDataRow）代表数据表里的一行数据，有自身的ID与对应数据文本解析方法

3.数据表（DataTable<T>）继承DataTableBase并实现了IDataTable<T>接口，使用字典来维护该表的所有数据行，并提供相关操作

4.数据表辅助器（IDataTableHelper）负责处理数据表文本（加载，切割，释放）

5.数据表管理器（DataTableManager）使用字典维护所有IDataTable，持有一个IDataTableHelper的引用，通过调用IDataTableHelper里的方法提供相关操作。持有一个IResourceManager，通过其进行数据表的加载（**加载过程中通过4个委托将资源加载事件通知给ResourceComponent，然后由其派发4个全局事件**）

6.加载数据表事件分为两部分，一部分是DataTableManager里的模块局部事件，定义在GF里，负责将资源加载事件通知到DataTableComponent，另一部分是定义在UGF里的全局事件，由DataTableComponent接收到资源加载事件后进行派发，

加载数据表流程：

外部调用DataTableManager里加载数据表的方法，在该方法里调用ResourceManager的异步加载资源方法

ResourceManager在异步加载资源成功后调用DataTableManager的回调方法，回调方法里调用DataTableHelper的加载数据表方法

DataTableHelper在加载数据表方法里调用DataTableManager的创建数据表方法

DataTableManager在创建数据表方法里调用DataTableHelper的分割数据表文本方法，得到的分割后的数据行，然后创建数据表对象，将数据行文本依次添加到数据表对象中，最后将数据表对象放入字典，至此加载完毕

DataTableManager.LoadDataTable()→ResourceManager.LoadAsset()→DataTableManager.LoadDataTableSuccessCallback()→DataTableHelper.LoadDataTable()→DataTableManager.CreateDataTable→DataTableHelper.SplitToDataRows()

Config（配置）

Config模块与DataTable模块类似，区别在于配置模块无需建立不同的数据行类

1.Config模块主要由配置数据，配置辅助器，配置管理器，加载配置事件四个部分组成

2.配置数据（ConfigData）保存配置项的数据，有bool，int，float，string四种类型

3.配置辅助器（IConfigHelper）负责进行配置的实际操作（加载，解析，释放）

4.配置管理器（ConfigManager）使用一个字典来维护所有ConfigData，持有一个IConfigHelper的引用，通过调用IConfigHelper里的方法提供相关操作。持有一个IResourceManager，通过其进行配置的加载（**加载过程中通过4个委托将配置加载事件通知给ConfigComponent，然后由其派发4个全局事件**）

5.加载配置事件分为两部分，一部分是ConfigManager里的模块局部事件，定义在GF里，负责将资源加载事件通知到ConfigComponent，另一部分是定义在UGF里的全局事件，由ConfigComponent接收到配置加载事件后进行派发，

配置加载流程：

外部调用ConfigManager里加载配置的方法，在该方法里调用ResourceManager的异步加载资源方法

ResourceManager在异步加载资源成功后调用ConfigManager的回调方法，回调方法里调用ConfigHelper的加载配置方法

ConfigHelper在加载配置方法里调用ConfigManager的解析配置方法

ConfigManager在解析配置方法里调用ConfigHelper的解析配置方法

ConfigConfigHelper在解析配置方法里解析配置文本，然后调用ConfigManager的添加配置方法，将配置数据添加到字典里，至此加载完毕

ConfigManager.LoadConfig()→ResourceManager.LoadAsset()→ConfigManager.LoadConfigSuccessCallback()→ConfigHelper.LoadConfig()→ConfigManager.ParseConfig()\*→ConfigHelper.ParseConfig()→ConfigManager.AddConfig()

Localization（本地化）

Localization模块提供本地化功能，也就是我们平时所说的多语言。Game Framework 在本地化方面，不但支持文本的本地化，还支持任意资源的本地化，

1.Localization模块主要由本地化辅助器，本地化管理器，加载字典事件三部分组成

2.本地化辅助器（ILocalizationHelper）负责进行字典的实际操作（加载，解析，释放）

3.本地化管理器（LocalizationManager）使用一个字典来维护本地化数据，持有一个ILocalizationHelper的引用，通过调用ILocalizationHelper里的方法提供相关操作。持有一个IResourceManager，通过其进行字典的加载（**加载过程中通过4个委托将字典加载事件通知给LocalizationComponent，然后由其派发4个全局事件**）

4.加载字典事件分为两部分，一部分是LocalizationManager里的模块局部事件，定义在GF里，负责将资源加载事件通知到LocalizationComponent，另一部分是定义在UGF里的全局事件，由LocalizationComponent接收到加载字典事件后进行派发，

Scene（场景）

Scene模块提供场景管理的功能，可以同时加载多个场景，也可以随时卸载任何一个场景，从而很容易地实现场景的分部加载。

1.Scene模块主要由场景管理器，场景事件两部分组成

2.场景管理器（SceneManager）分别使用3个列表维护已加载，加载中，已卸载的场景，持有IResourceManager引用，调用其中的加载场景与卸载场景方法，持有一个IResourceManager，通过其进行字场景的加载与卸载（**加载与卸载过程中通过6个委托将场景加载与卸载事件通知给SceneComponent，然后由其派发6个全局事件**）

3.场景事件分为两部分，一部分是SceneManager里的模块局部事件，定义在GF里，负责将场景加载与卸载事件通知到SceneComponent，另一部分是定义在UGF里的全局事件，由SceneComponent接收到加载与卸载事件后进行派发

Entity（实体）

GF将游戏场景中，动态创建的一切物体定义为实体。

Entity模块提供管理实体和实体组的功能，如显示隐藏实体、挂接实体（如挂接武器、坐骑，或者抓起另一个实体）等。实体使用结束后可以不立刻销毁，从而等待下一次重新使用。

1.Entity模块主要由实体，实体信息，实体组，辅助器，实体管理器，实体事件五个部分组成

2.实体（IEntity）保存了实体相关数据以及方法

3.实体信息（EntityInfo）主要保存实体的父子实体引用

4.实体组（EntityGroup）负责使用对象池来管理实体

5.辅助器分为实体辅助器（IEntityHelper）与实体组辅助器（IEntityGroupHelper），IEntityHelper提供了实体的实例化，创建与释放的方法，IEntityGroupHelper主要在UGF中为实例化出来的实体提供默认父实体

6.实体管理器（EntityManager）使用字典管理所有EntityInfo与EntityGroup，持有一个IEntityHelper引用，通过其中的方法进行实体相关的操作。持有一个IResourceManager，通过其进行实体资源的加载（**加载过程中通过4个委托将实体资源加载事件通知给EntityComponent，然后由其派发4个全局事件**）。

7.实体事件分为两部分，一部分是EntityManager里的模块局部事件，定义在GF里，负责将实体资源加载事件与隐藏实体事件通知到EntityComponent，另一部分是定义在UGF里的全局事件，由EntityComponent接收到事件后进行派发

8.在UGF中，还提供了继承MonoBehaviour的EntityLogic（实体逻辑基类）与Entity（实体），EntityLogic负责处理实体的逻辑（实体的初始化，显示，隐藏，子实体的附加，解除），Entity实现了IEntity接口，持有并代理对应的EntityLogic

9.StartForce中，提供了EntityData（实体数据基类），由继承了EntityLogic的类持有，并在OnShow方法中赋值

UI（界面）

UI模块提供管理界面和界面组的功能，如显示隐藏界面、激活界面、改变界面层级等。不论是 Unity 内置的 uGUI 还是其它类型的 UI 插件（如 NGUI），只要派生自 UIFormLogic 类并实现自己的界面类即可使用

1.UI模块主要由界面，界面信息，界面组，辅助器，界面管理器，界面事件五个部分组成

2.界面（IUIForm）保存了界面相关数据以及方法

3.界面信息（UIFormInfo）主要保存界面的引用以及设置界面是否暂停或遮挡

4.界面组（UIGroup）使用链表维护组内所有UIFormInfo

5.辅助器分为界面辅助器（IUIFormHelper）与界面组辅助器（IUIGroupHelper），IUIFormHelper提供了界面的实例化，创建与释放的方法，IUIGroupHelper主要在UGF中为实例化出来的界面提供默认父实体

6.界面管理器（UIManager）使用字典管理所有UIGroup，使用链表管理所有IUIForm，并负责使用对象池来管理某界面。持有一个IUIFormHelper引用，通过其中的方法进行界面相关的操作。持有一个IResourceManager，通过其进行界面资源的加载（**加载过程中通过4个委托将实体资源加载事件通知给UIComponent，然后由其派发4个全局事件**）。

7.界面事件分为两部分，一部分是UIManager里的模块局部事件，定义在GF里，负责将界面资源加载事件与关闭界面事件通知到UIComponent，另一部分是定义在UGF里的全局事件，由UIComponent接收到模块局部事件后进行派发

8.在UGF中，还提供了继承MonoBehaviour的UIFormLogic（界面逻辑基类）与UIForm（界面），UIFormLogic负责处理界面的逻辑（界面的初始化，打开，关闭，暂停，激活），UIForm实现了IUIForm接口，持有并代理对应的UIFormLogic

Sound（声音）

Sound模块提供管理声音和声音组的功能，用户可以自定义一个声音的音量、是2D声音还是3D声音，甚至是直接绑定到某个实体上跟随实体移动。

1.Sound模块主要由辅助器，声音代理，声音组，声音管理器，声音事件五个部分组成

2.辅助器分为声音代理辅助器（ISoundAgentHelper），声音组辅助器（ISoundGroupHelper），声音辅助器（ISoundHelper）。ISoundAgentHelper负责提供对声音的各种实际操作（声音的播放，暂停，恢复等），ISoundHelper负责提供声音的释放操作

1. 声音代理（SoundAgent）持有一个ISoundAgentHelper引用，并通过调用其中的方法进行声音的操作

4.声音组（SoundGroup）使用列表维护该组所有SoundAgent，并提供声音的相关操作（播放，暂停，恢复等）

5.声音管理器（SoundManager）使用字典管理所有SoundGroup，持有一个IResourceManager，通过其进行声音资源的加载（**加载过程中通过4个委托将声音资源加载事件通知给SoundComponent，然后由其派发4个全局事件**）。

6.声音事件分为三部分，一部分是SoundManager里的事件，另一部分是SoundAgentHelper里的事件，这两部分定义在GF里，是**模块局部事件**，第三部分则是**UGF里定义的全局事件**,由SoundComponent接收到模块局部事件后进行派发

Network（网络）

Network模块提供使用 Socket 长连接的功能，支持 TCP 协议，同时兼容 IPv4 和 IPv6 两个版本。

1.Network模块主要由消息包、网络频道辅助器、网络频道、网络管理器、网络事件五个部分组成

2.消息包分为消息包基类（Packet）【继承BaseEventArgs，这意味着消息包在GF里也是一类全局事件】、消息包头接口（IPacketHead）、消息包处理器接口（IPacketHandler）。如果想接入 ProtoBuf 之类的协议库，只要派生自 Packet 类并实现自己的消息包类即可使用

3.网络频道辅助器（INetworkChannelHelper）主要负责消息包的序列化与反序列化

4.网络频道（NetworkChannel）持有Socket的引用进行网络连接、消息发送、消息接收，持有INetworkChannelHelper的引用进行对消息包的操作，使用队列管理要发送的消息包，使用事件池来处理接收到的消息包（进行消息包派发）（**通过5个委托向NetworkManager通知网络事件**）

网络连接流程：Connect()【检查协议族→创建Socket对象→开始异步连接】→连接完毕，调用回调→ ConnectCallback()【结束连接→为接收网络消息包作准备→重置心跳包状态→开始接收消息】

消息接收流程：Receive()【异步接收消息】→接收完毕，调用回调→ReceiveCallback()【计算接收到的消息字节数量→计算流位置→处理接收到的消息包】→ProcessPacketHeader()【反序列化消息包头→为接收消息包作准备】 ProcessPacket()【重置心跳包状态→反序列化消息包→派发消息包→为接收消息包头作准备】

消息发送流程：Send<T>()【将要发送的网络消息包加入队列】→在Update中调用消息包发送处理方法→ProcessSend()【从队列里获取要发送的消息包对象→序列化消息包→调用发送消息的方法】→Send()【异步发送消息】→发送完毕，调用回调→SendCallback()【计算偏移→重置发送状态】

5.网络管理器（NetworkManager）使用字典维护所有NetworkChannel，提供对NetworkChannel相关的操作（**通过5个委托向NetworkComponent通知网络事件，然后由其派发5个全局事件**）

6.网络事件分为两部分，一部分是NetworkManager里的模块局部事件，定义在GF里，负责将网络事件通知到NetwrokComponent，另一部分是定义在UGF里的全局事件，由NetworkComponent接收到模块局部事件后进行派发

7.在StartForce中接入了ProteBuf，实现了INetworkChannelHelper、Packet、IPacketHandler、IPacketHeader