**AD.BASE AD\_I Interface**

# 双类转换接口

IBase以及IBaseMap通过To/From函数互相转换，IBase应被用于运行时的实体或数据，IBaseMap应被用于保存数据时提供最佳的支持

两个接口都有对应的泛型版本，意味着更加明确的目标和严格的约束

**ToMap** 将IBase自身转换为IBaseMap

**FromMap** 以IBaseMap的内容初始化IBase的内容

**ToObject** 将IBaseMap自身转换为IBase

**FromObject** 以IBase的内容初始化IBaseMap的数据

**Serialize** 将IBaseMap转换为可保存的字符串，用于Deserialize解析

**Desrialize** 从源字符串解析出应用于自身的数据，其应为Serialize生成

**AD.BASE AD\_S Implementation**

# AD异常类

**ADExcption**会在生成的同时记录事件，作为AD中常用的异常类型

除此之外没有特别特殊的特点

**NullArchitecture**代表没有注册进架构或对应架构已被销毁

# AD架构接口

**IAnyArchitecture** 所有AD架构接口的最底层，并且不具备任何内容和实现

**ICanInitialize** 拥有一个**Init**函数，默认用于初始化或重置类

**ICanGetArchitecture** （旧版本）通过两个函数设置或获取**IADArchitecture**目标

（目前推荐使用）Architecture {get; set; }

**ICanGetSystem** 实现以支持获取架构中System

**ICanGetModel** 实现以支持获取架构中Model

**ICanGetController** 实现以支持获取架构中Controller

**ICanSendCommand** 实现以支持在架构中发送Command

**ICanMonitorCommand**

实现以支持在Command Diffusing过程中激活**OnCommandCall函数**

**IADModel** 实现以支持在架构中作为数据类

**IADSystem** 实现以支持在架构中作为系统类

**IADController** 实现以支持在架构中作为控制器类

**IADCommand** 实现以支持在架构中作为命令类

**IADMessage** 被用于架构内收集消息，具有一个返回字符串的**What**函数

# AD架构标准实现

**ADModel** 实现**IADModel**并提供了默认的实现

**ADSystem** 实现**IADSystem**并提供了默认的实现

**MonoSystem** 实现**IADSystem**并继承了**MonoBehaviour**

**ADController**

实现**IADController**并继承了**MonoBehaviour**,能够在OnDestroy（保护的虚函数）中将自身解除在架构中的注册关系

**ADCommand** 实现**IADCommand**并提供了默认的实现

**Vibration** 继承**ADCommand**并用于Command Diffusing

**ADMessage** 实现**IADMessage**并提供了默认的实现，记录生成的时间

**ADMessageRecord**

一个默认的消息收集器，在默认的**ADArchitecture**实现中被使用，实现了**IADModel**

AD.BASE Event from Unity & ExtAD

# Invokable Call触发器

**ADBaseInvokableCall**

抽象类，提供了Invokable Call体系的通用实现，能够得到执行的函数或委托的相关消息，在触发时会记录Debug信息

**template<typename T...> ADInvokableCall**

继承了**ADBaseInvokableCall的**泛型（最多支持四个泛型参数）

**ADBaseOrderlyEvent**

抽象类，其中**Invoke(params object[] args)**函数的参数，在**ADOrderlyEvent**的实现中类型不正确将尝试使用**default**填充

**template<typename T...> ADOrderlyEvent**

继承了**ADBaseOrderEvent**的泛型（最多支持四个泛型参数），提供了有序执行委托或函数的能力，并能够获取其中相关的真实信息

**template<typename T...> ADEvent**

继承了对应泛型参数的UnityEvent（最多支持四个泛型参数），在使用**Invoke**触发错误时会产生与**ADBaseOrderlyEvent**相同的日志报告，在一切正常情况下将不会产生额外的行为，你可以通过**宏ADEVENT\_DISABLE\_TRY**使得**ADEvent**关闭try-catch以降低性能上的损耗或避免意外的消息收集

AD.BASE Property Get/Set Binding

# IPropertyHasGet/Set & PropertyExtension

有关绑定实体，依赖于扩展函数以实现多态，功能由接口的扩展函数直接提供，在实现接口后即可直接使用而无需自行实现细节（你依然可以通过继承以指定新的函数）

**IPropertyHasGet** 只读绑定器

**IPropertyHasSet** 只写绑定器

以上两个接口必须指定目标类型作为第一个泛型参数，可选指定使用的**PropertyAssets**作为第二个泛型参数，该类型将用于保存目标实例，默认为**基类PropertyAsset**

**PropertyAssets** 保存对象的实例

**Property** 封装**PropertyAssets**

**AbstractBindProperty** 抽象类，指定绑定行为，并支持共享**Property**，伪指针

**默认提供的BindProperty系列**

该部分用于提供立即使用的具有简单行为的绑定器

**PropertyExtension** 静态类，对接口的功能进行实现，内置了**BindProperty等**的更多功能与类型

**\*ValueToStringPropertyAsset**

**\*StringToValuePropertyAsset**

这两个类是能够将string与float互相转换的**PropertyAsset**实现

AD.BASE Object Extension

**IInvariant** 抗逆变性，在**As等**AD实现的扩展函数中将拒绝转变为目标类型

**[ObjectExtension]**

**AllArchitecture ADArchitecture**的实现在生成instance时会进入该字典

**As** 尝试转换为目标类型

**Convertible** 是否可以转换为目标类型

**Is** 是否可以转换为目标类型

**IsAssignableFromOrSubClass** 两个类型之间是否具有继承关系

**ClassCorrelation** 指示类型之间的继承关系

**DetectCorrelation** 检测两个类型间的继承关系并返回**ClassCorrelation**

**PrefabInstantiate** 预制体实例化

**ObtainComponent** 通过简单的GetComponents获取目标类型的脚本

**Fetch & Share** 管道式声明变量引用

**SeekComponent**

获取能够被转换为目标类型的脚本，推荐程度超过**ObtainComponent**