**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

|  |  |
| --- | --- |
| public ADException() | 构造函数 |
| public ADException(string) | 带消息的构造函数 |
| public ADException(string, Exception) | 带消息和源异常的构造函数 |
| public ADException(Exception) | 带源异常的构造函数 |
| protected ADException(SerializationInfo,StreamingContext) | 带序列化信息的构造函数 |
| public string Serialize() | 返回错误消息，在开头会标记时间 |
| public string SerializeStackTrace() | 返回错误帧栈消息，在开头会标记时间 |
| public string SerializeSource() | 返回错误源，在开头会标记时间 |
| public string SerializeMessage() | 同Serialize |
| public string SerializeHelpLink() | 返回HelpLink文本，会在开头标记时间 |
| private DateTime AD\_\_GeneratedTime | 记录生成时的时间 |

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

|  |  |
| --- | --- |
| public NullArchitecture() | 构造函数 |

# 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

|  |  |
| --- | --- |
| void Init | 初始化 |

**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; }

|  |  |
| --- | --- |
| IADArchitecture ADInstance() | 获取架构（已过时） |
| void SetArchitecture(IADArchitecture) | 设置架构（已过时） |
| IADArchitecture Architecture { get; set; } | 架构 |

**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**

|  |  |
| --- | --- |
| void OnCommandCall(\_Command) | 当对应的Command在架构内传播时触发 |

**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

|  |  |
| --- | --- |
| void Execute() | 命令被执行时调用的函数 |
| string LogMessage() | 被执行时生成的信息 |

以上四个接口均实现ICanGetArchitecture

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

**IADArchitecture** 架构接口

|  |  |
| --- | --- |
| IADArchitecture AddMessage(string) | 记录消息 |
| \_Model GetModel<\_Model>() | 获取数据类 |
| \_System GetSystem<\_System>() | 获取系统类 |
| \_Controller GetController<\_Controller>() | 获取控制器 |
| IADArchitecture RegisterModel<\_Model>(\_Model) | 注册数据类 |
| IADArchitecture RegisterSystem<\_System>(\_System) | 注册系统类 |
| IADArchitecture RegisterController<\_Controller>(\_Controller) | 注册控制器 |
| IADArchitecture RegisterCommand<\_Command>(\_Command) | 注册命令 |
| IADArchitecture RegisterModel<\_Model>() | 注册数据类 |
| IADArchitecture RegisterSystem<\_System>() | 注册系统类 |
| IADArchitecture RegisterController<\_Controller>() | 注册控制器 |
| IADArchitecture RegisterCommand<\_Command>() | 注册命令 |
| public IADArchitecture SendImmediatelyCommand<\_Command>() | 立即执行命令 |
| public IADArchitecture SendImmediatelyCommand<\_Command>(\_Command command) | 立即执行命令 |
| IADArchitecture SendCommand<\_Command>() | 发送并执行命令 |
| void Diffusing<\_Command>() | 对目标命令执行扩散 |
| void Diffusing<\_Command>(\_Command command) | 对目标命令执行扩散 |
| void Send<\_Command, \_CanMonitorCommand>() | 向指定的监视类型发送命令并使其执行 |
| void Send<\_Command, \_CanMonitorCommand>(\_Command) | 向指定的监听类型发送该命令并使其执行 |
| IADArchitecture UnRegister<\_T>() | 解除注册 |
| IADArchitecture UnRegister(Type type) | 解除注册 |
| IADArchitecture Register<\_T>() | 注册 |
| IADArchitecture Register<\_T>(\_T) | 注册 |
| bool Contains<\_Type>() | 是否包含 |

# 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

|  |  |
| --- | --- |
| protected virtual void OnDestroy() | 销毁时会自动解除注册 |

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

|  |  |
| --- | --- |
| public abstract void OnExecute() | 被嵌套在Execute中，在Execute中先添加本命令执行的消息后调用该函数 |

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

Vibration中的Execute是空函数，其仅为Command Diffusing进行专业的支持

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

|  |  |
| --- | --- |
| public ADMessage(string) | 构造函数 |
| public string AD\_\_Message | 消息 |
| public string What() | 返回构造的消息 |

**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**

|  |  |
| --- | --- |
| public string What() | 返回构造的消息 |
| public void Add(IADMessage) | 新增消息 |
| public void Remove(IADMessage) | 移除消息 |
| public virtual void Save(string) | 于指定路径保存 |
| public int Count.get | 获取当前消息总数 |
| public int MaxCount | 设置消息上限 |
| private List<IADMessage> AD\_\_messages | 消息列表 |

**ADArchitecture** 默认的架构实现，非常重要的类型，以下称呼的目标类型有些是对实例收集器**AD\_\_Objects**中键的称呼

|  |  |
| --- | --- |
| protected ADArchitecture() | 构造函数 |
| public static void Destory() | 立即清除**\_\_ADinstance**与默认实现内的有关引用，此时其他仍引用其实例的对象再调用架构实例可能发生未知的错误 |
| public static T instance | 获取架构单例，若不存在则会立刻生成同时调用其Init函数进行初始化，并加入**ObjectExtension.AllArchitecture**中 |
| protected ADMessageRecord MessageRecord | 获取本架构的消息收集器 |
| protected virtual object Get<\_T>() | 查找目标类型的实例 |
| public IADArchitecture UnRegister(Type) | 解除目标类型的注册 |
| public IADArchitecture UnRegister<\_T>() | 解除目标类型的注册 |
| public virtual void SaveRecord() | 保存消息 |
| public virtual void Init() | 初始化 |
| public IADArchitecture Register<\_T>(\_T) | 以实例注册 |
| public IADArchitecture Register<\_T>() | 以new()注册 |
| public bool Contains<\_Type>() | 查询目标类型是否已注册，注，已注册并不一定实例存在，其可能被GC强制回收等手段在注册后销毁 |
| public virtual IADArchitecture AddMessage(string) | 新增消息 |
| public IADArchitecture SendCommand<\_Command>() | 发送并执行命令 |
| public IADArchitecture SendImmediatelyCommand<\_Command>() | 立即发送并执行命令，若目标类型未注册则进行注册，若已注册则发送已注册命令实例 |
| public IADArchitecture SendImmediatelyCommand<\_Command>(\_Command) | 立即发送并执行命令，覆盖已有的目标类型实例 |
| public void Diffusing<\_Command>() &  public void Diffusing<\_Command>(\_Command) | **Command Diffusing**，将历遍全部已注册实体，若其是对应的**ICanMonitorCommand**的实现则调用其**OnCammandCall**。若目标类型未注册，则生成一个默认的命令（该命令不会自动注册进架构，在扩散过程中也不会调用**Execute**） |
| public void Send<\_Command, \_CanMonitorCommand>(\_Command) &  public void Send<\_Command, \_CanMonitorCommand>() | 发送命令，但不执行，**Command Diffusing**，效果同上，但不同的是会忽略抗逆变性**IInvariant** |
| **……本处忽略部分简单实现，如注册类型等** | **注册实例初始化函数调用晚于设置架构，因此在其Init函数中可以使用Architecture来获取架构实例** |
| private static IADArchitecture \_\_ADinstance | 架构单例 |
| private Dictionary<Type, object> AD\_\_Objects | 架构内的组件由此字典收集 |
| private object \_p\_last\_object | 在查找时会对比类型，若与上次查找的对象相同将返回该对象而跳过字典的查找，若否，则会在查询后重置该引用为新的目标 |
| private Type \_p\_last\_type | 保存上一次查找时的类型 |
| private List<Type> \_p\_null\_type | 若查找时未能找到以目标类型注册的实例，将被记录在此收集器中，在该目标类型真正注册时发出警告，以便用户检查逻辑与函数的调用顺序 |

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；

可以在检视面板上观察和调整其上调用的函数

**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，一般更推荐使用**SeekComponent**

**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，在使用**ObtainComponent**时一般更推荐使用这个方法

AD.BASE Event System Handler

**IADEventSystemHandler** 继承**IEventSystemHandler**，用法同

**ADEventSystemExtension** 静态类，用于触发事件

AD.BASE Debug Extension

全局Debug，比架构与ADG的消息收集更自动化，路径**LogPath**是公共字段，初始化为永久缓存路径根目录下的**Debug.dat**文件，格式为文本

**LogMethodEnabled** 默认开启，用于指示**Log**与**LogMessage**函数是否启用

**Log & LogMessage** 打印消息并记录当前函数帧栈

**GetStackTraceModelName** 获取当前的函数帧栈