UIFramework使用介绍

目录

[1 UI生命周期 2](#_Toc5587)

[1.1 OnAwake 2](#_Toc18998)

[1.2 OnStart 2](#_Toc31143)

[1.3 OnEnable 2](#_Toc15993)

[1.4 OnDisable 2](#_Toc3016)

[1.5 1.5 OnDestroy 2](#_Toc20730)

[1.6 1.6 OnNotifiy 2](#_Toc14514)

[2 UI类型 2](#_Toc24458)

[3 UIManager 3](#_Toc1334)

[3.1 UIManager.Instance.Register 3](#_Toc26891)

[3.2 UIManager.Instance.Push 3](#_Toc5295)

[3.3 UIManager.Instance.Pop 3](#_Toc19275)

[3.4 UIManager.Instance.Clear 3](#_Toc32274)

[4 GameEventManager 3](#_Toc6235)

[4.1 GameEventManager.Instance.RegistEvent 3](#_Toc20049)

[4.2 GameEventManager.Instance.RemoveEvent 3](#_Toc2026)

[4.3 GameEventManager.Instance.Notify 4](#_Toc19168)

[5 UI点击事件注册 4](#_Toc27694)

[6 UI动画 4](#_Toc23906)

[6.1 Enable状态 4](#_Toc3935)

[6.2 Disable状态 6](#_Toc6602)

1. UI生命周期

1、OnAwake

2、OnStart

3、OnEnable

4、OnDisable

5、OnDestroy

6、OnNotifiy

* 1. OnAwake

UI加载完后会执行OnAwake，该函数只会执行一次，该函数适合UI控件的初始化、音效初始化，以及一些耗时的加载会放到该函数去做，别访问数据相关的东西

* 1. OnStart

UI显示后只执行一次（注意：从栈中激活并不会执行）。 该函数可以接收外部传参 如：UIManager.Instance.Open(“xx”,args)，args的参数就会传到了OnStart(args)

* 1. OnEnable

执行OnStart后会执行OnEnable，栈中UI从新激活也会调用OnEnable，该函数和OnDisable是一对的，建议UI的显示都可以在该函数处理

* 1. OnDisable

UI每次隐藏都会调用，OnDestroy之前也会先调用OnDisable，隐藏需要释放的东西可以在该函数处理

* 1. 1.5 OnDestroy

UI销毁会调用，前提是UI执行过OnAwake，建议UI加载的资源都要在该函数处理释放

* 1. 1.6 OnNotify

接收 GameEventManager.Instance.Notify(evt,args) 发送的通知

1. UI类型
2. Normal (栈)
3. Popup （栈）

2、Dialog

3、Guide

4、Tips

5、TopMask

6、Top

目前类型Normal、Popup是用栈管理的，而且UI可以被重复创建（为了实现记录之前ui的状态），其他类型的UI只能有一个实例。越往下的类型会覆盖之前类型的UI

1. UIManager

管理游戏所有的UI，全部UI的显示都会添加到UIType对应的栈管理器

* 1. UIManager.Instance.Register

所有UI都必须使用该函数注册，注册后才能正常使用。

函数参数说明:

uiName UI名字

uiType UI类型

uiResType UI加载方式

hasAnimation UI是否有动画

C#： 使用UIAttribute进行注册

[UIAttribute("MainMenuUI", UIType.Normal, UIResType.Resorces, true)]

Lua: LuaUIManager.Register(LuaUI,uiName,uiType ,uiResType ,hasAnimation)

* 1. UIManager.Instance.Open(uiName,args)

打开指定名字的UI，并把args参数传递到OnStart()

* 1. UIManager.Instance.OpenWithCallback(uiName,callback,ar gs)

打开指定名字UI，并把args参数传递到OnStart(),打开后执行回调

* 1. UIManager.Instance.Close(uiName)

关闭指定UI

* 1. UIManager.Instance.CloseWithCallback(uiName,callback)

关闭指定UI,关闭完成后有回调

* 1. UIManager.Instance.Clear()

关闭所所有UI

* 1. UIManager.Instance.PopThenOpen(uiName,args)

前一个UI退栈，并打开一个新的ui，把args参数传递到新打开的ui 的OnStart

* 1. UIManager.Instance.PopAllThenOpen(uiName,args)

Noraml层级全部UI退栈，并打开新的ui，并把args传递到新打开的 UI的OnStart

* 1. UIManager.Instance.Remove(uiName)

移除所有名字为uiName的UI，不会播放ui退场动画

* 1. UIManager.Instance.RemoveOne(uiName)

从UI容器删除一个指定名字的一个UI，只会删一个，如果出现多个也只是删一个

1. GameEventManager

通知管理器，可以同时给C#、Lua、ILRT派发通知

* 1. GameEventManager.Instance.RegistEvent

注册事件监听：

C#:

GameEventManager.Instance.RegistEvent(IBaseEventListener listener)

Lua:

GameEventManager.Instance.RegistEvent(new LuaProxy(()=>{}))

* 1. GameEventManager.Instance.RemoveEvent

移除事件通知

* 1. GameEventManager.Instance.Notify

派发事件

如：GameEventManager.Instance.Notify(“evt”,1,2,3,4)

1. UI点击事件注册

C#：

UI从UIMonoProxy继承

RegisterListener("Button", OnClickExitBtn) 或

RegisterListener(ButtonGameObject, OnClickExitBtn)

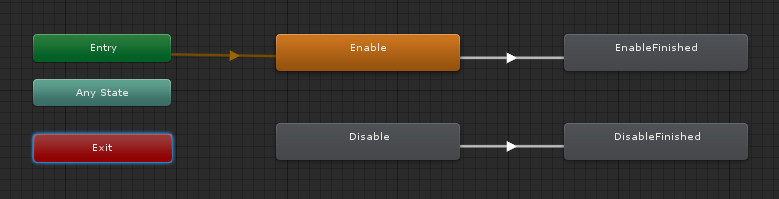
Lua:

UI从LuaUI继承

self:RegisterListener( "BackButton", self.OnClickExitBtn)

或  
 self:RegisterListener( BackButton, self.OnClickExitBtn)

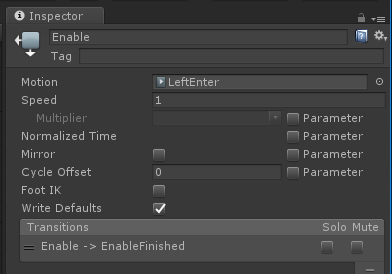
1. UI动画



* 1. Enable状态

Enable是UI的展示动画，可以指向一个自定义的clip。连线过度到EnableFinished，EnableFinished是完成的最后一个状态，一般停留在最后一帧

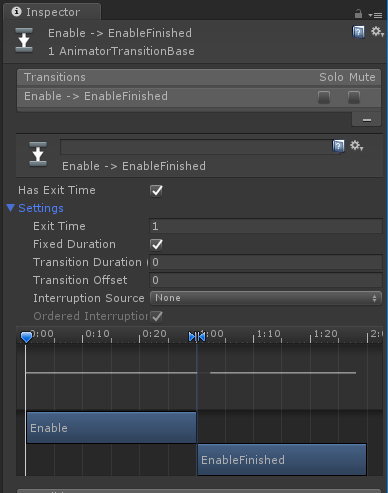
Enable状态设置：

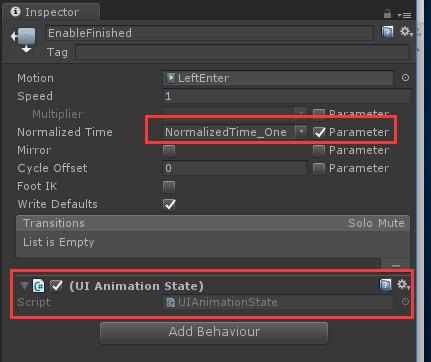


过度连线设置：

ExitTime = 1 说明上一个状态播放完才能切下一个状态

Trasnsition Duration = 0 过度时间要设置为0



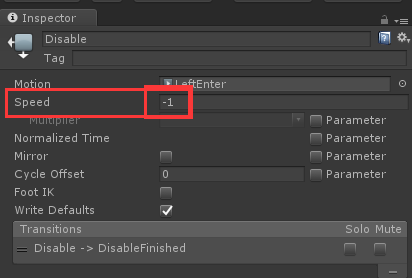
EnableFinished状态设置：  
 

* 1. Disable状态

Disable是UI的隐藏动画，可以指向一个自定义的clip。连线过度到DisableFinished，DisableFinished是隐藏的最后一个状态，一般停留在第0帧

Disable状态设置：

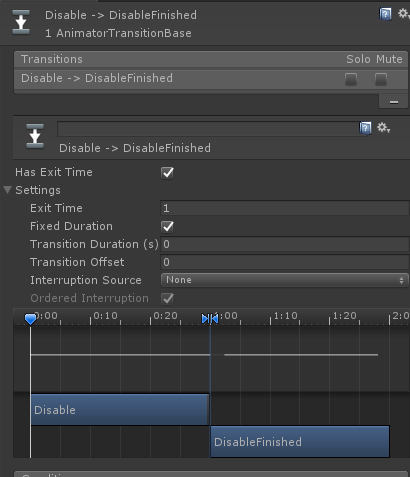
Speed = -1可以实现倒播，好处就是可以复用Enable的clip



过度连线设置：

ExitTime = 1 说明上一个状态播放完才能切下一个状态

Trasnsition Duration = 0 过度时间要设置为0



DisableFinished状态设置：

