## 现有UI管理器的缺陷

1. 界面的先后层级不是由调用EnterWindow的先后顺序决定，而是跟异步加载资源的速度有关，存在很多随机可能。 比如：

UIWindowManager.EnterWindow(ViewName1);

UIWindowManager.EnterWindow(ViewName2);

如果viewName2 是已经缓存的资源，或者异步加载时 viewNam2比viewName1提前加载好资源。 那么viewName1 将显示在 viewName2的前面。

解决办法如下：

我们给每个界面提前生成了 一个挂载点Transform（HintGo）。 界面资源异步加载完成后的GameObject实例直接 挂载到对应的HintGo。 界面之间的层级 由HintGo之间决定就可以了。 由于HintGo是同步实时生成的， 这样避免了异步的问题。

1. EnterWindow，ExitWindow无法随意组合调用。 比如：

UIWindowManager.EnterWindow(ViewName1);

UIWindowManager.ExitWindow(ViewName2);

解决办法：

增加一个被关闭界面的缓存，当ExitWindow的时候，只做移除UI栈的操作， UIWindow缓存在一个列表里。 具体关闭逻辑在Update里面处理，每帧判断UIWindow是否已经完全走完Enter流程， 只要走完Enter流程的才执行Exit流程。

(3) 界面动画的播放完成回调由 Animation决定。 而Animation只有GameObject.activeSelf==true时才会播放动画。 所以如果一个界面 处于隐藏状时，无法正确被ExitWindow。 比如：

UIWindowManager.EnterWindow(ViewName1);

UIWindowManager.EnterWindow(ViewName2);

UIWindowManager.ExitWindow(ViewName1).

ViewName1, ViweName2 两个界面都是全屏界面。 ViewName1会被ViewName2

盖住导致ViewName1界面 activeSelf==false。 这时候执行ExitWindow（ViewName1）

操作时， ViewName1无法被关闭掉。

解决办法：

界面动画播放还是由Animation负责 但界面动画播放完成的控制 由Timer定时器决定。 我们可以去Animation得到动画时长， 然后设置一个 定时器即可。

（4） 界面配置不明确，

## 重构后UI管理器的结构

### 界面配置：

新的UI管理器中，将新增一个Lua文件，用于集中填写 UI界面的配置。 UI界面的内容还是跟以前一样。

## 界面层级

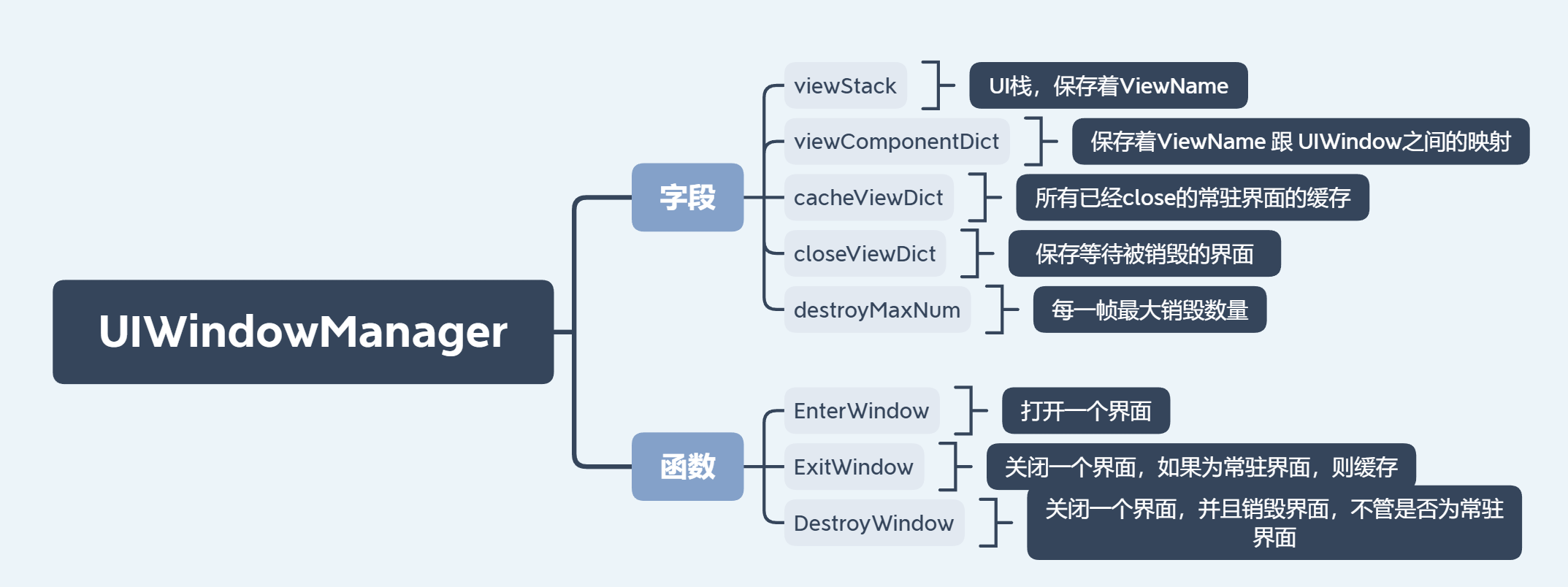
界面层级沿用以前。 采用以下几个层级：

## 界面类型

界面类型沿用以前。 采用以下两个类型：

## UIWindowManager修改

新的UIWindowManger的结构图如下：



viewStack：UI栈，按顺序保存着所有已经EnterWindow界面的ViewName。

viewCompDict：保存着所有已经打开界面中ViewName跟UIWindow的映射，可以根据一个ViewName快速找到对应UIWindow。

cacheViewDict：保存所有已经关闭的常驻界面。 如果常驻界面下次被EenterWindow，不需要再实例化资源，直接从这里取就可以了。

closeViewDict：保存等待被Close的界面集合。

destroyMaxNum：每一帧可销毁界面的最大数量。

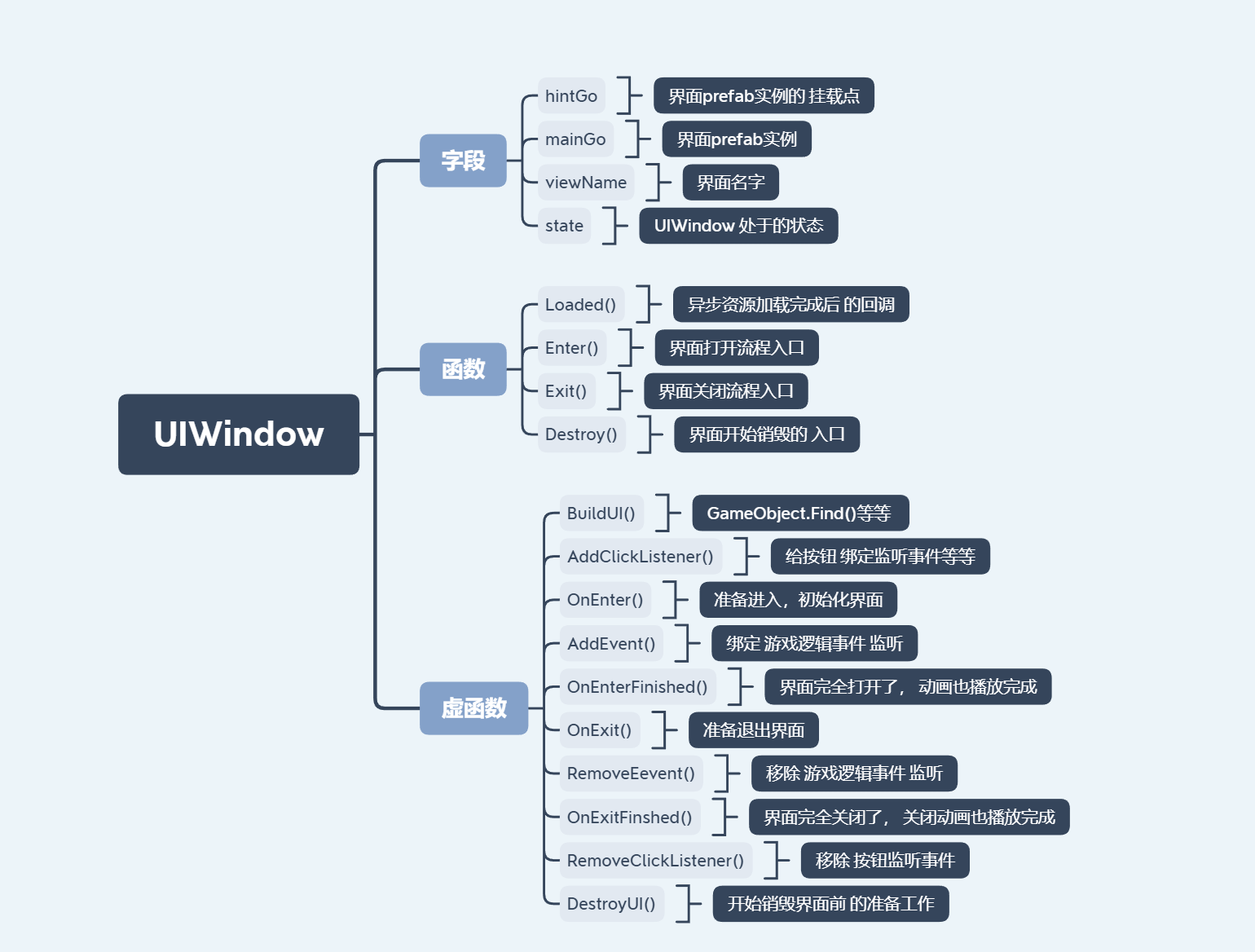
EnterWindow：打开一个界面的接口。

ExitWindow：关闭一个界面的接口。

DestroyWindow：关闭并销毁一个界面的接口。

Update：帧刷新函数，在这个函数里面，会遍历closeViewDict里的每一个界面，分别对每个待关闭界面执行 关闭逻辑。

## UIWindow修改

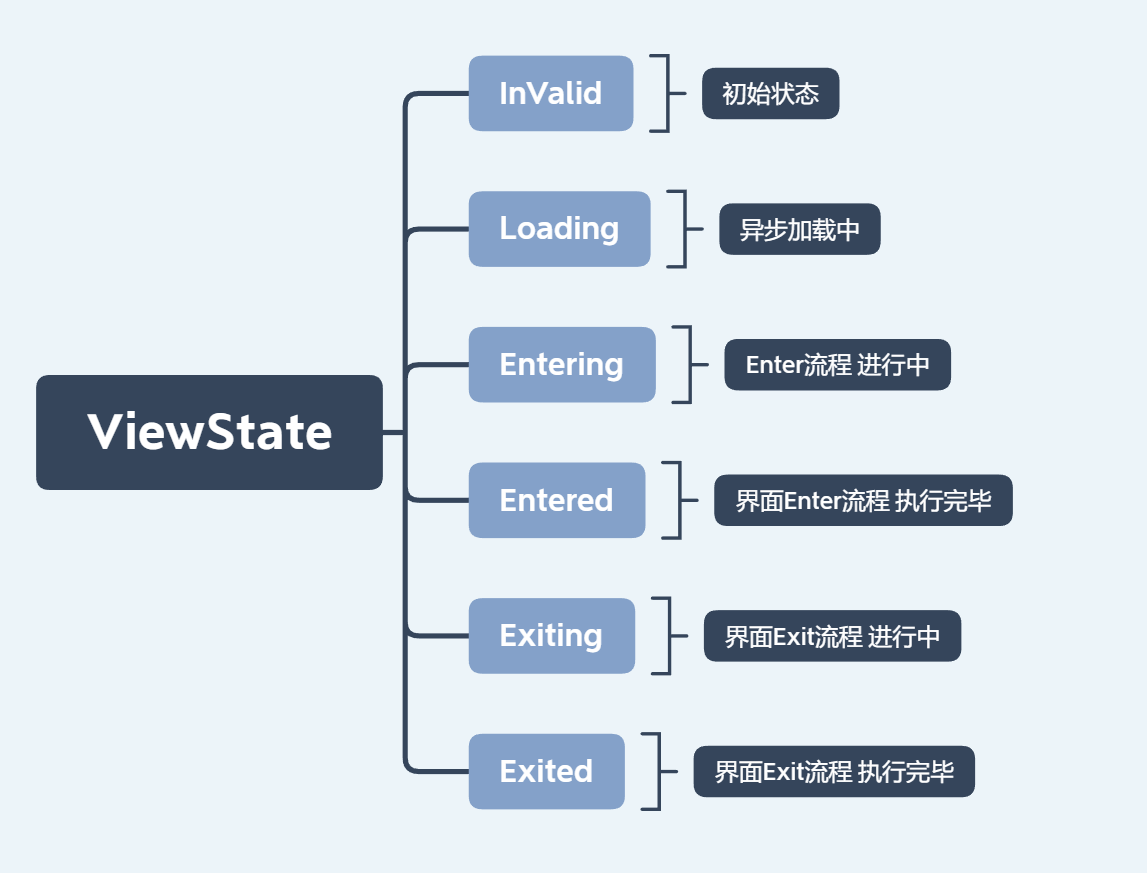


hintGo：界面prefab实例的挂载点。

mainGo：界面prefab实例。

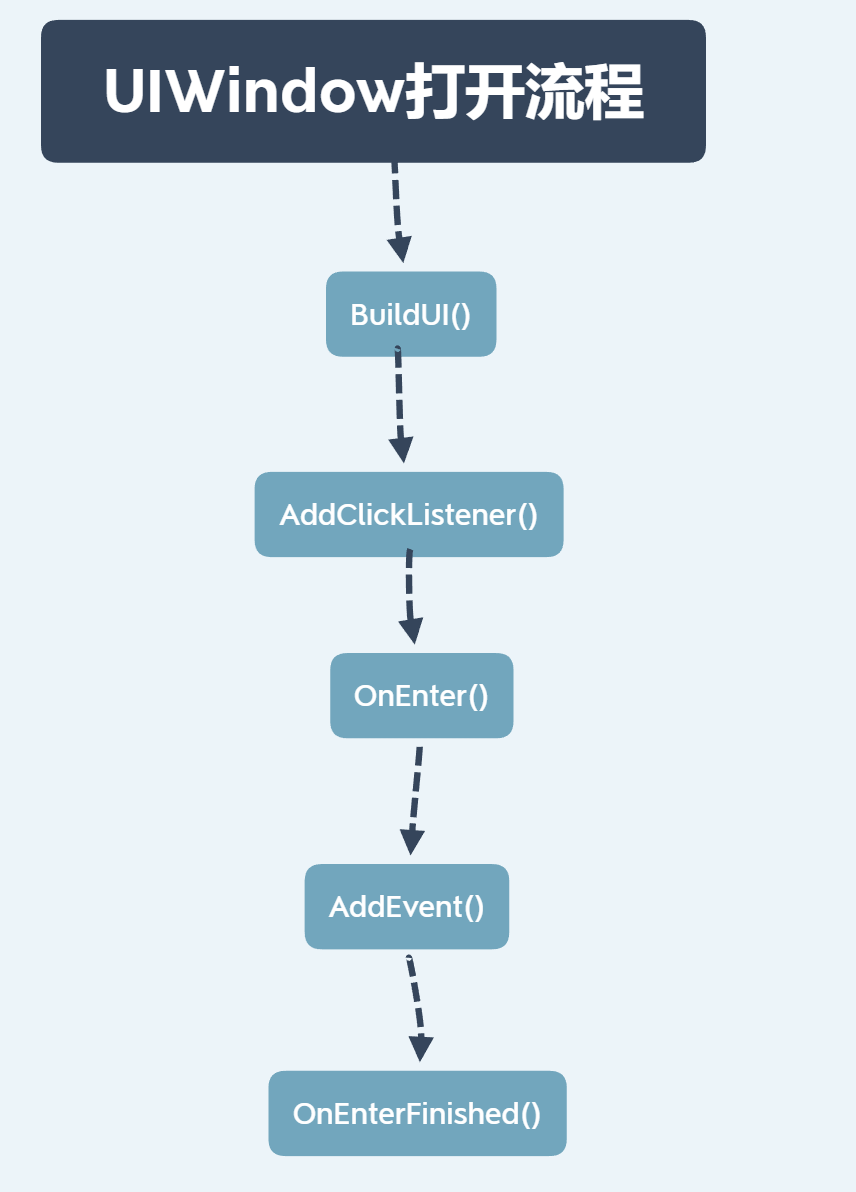
viewName：界面名字。

state：界面处于的状态。一共有一下几种：



虚函数部分定义的是 界面打开，关闭的各自流程。

具体打开流程的先后顺序如下：



**（1）BuildUI**：主要执行一些获取组件，获取GameObject等等初始化操作。

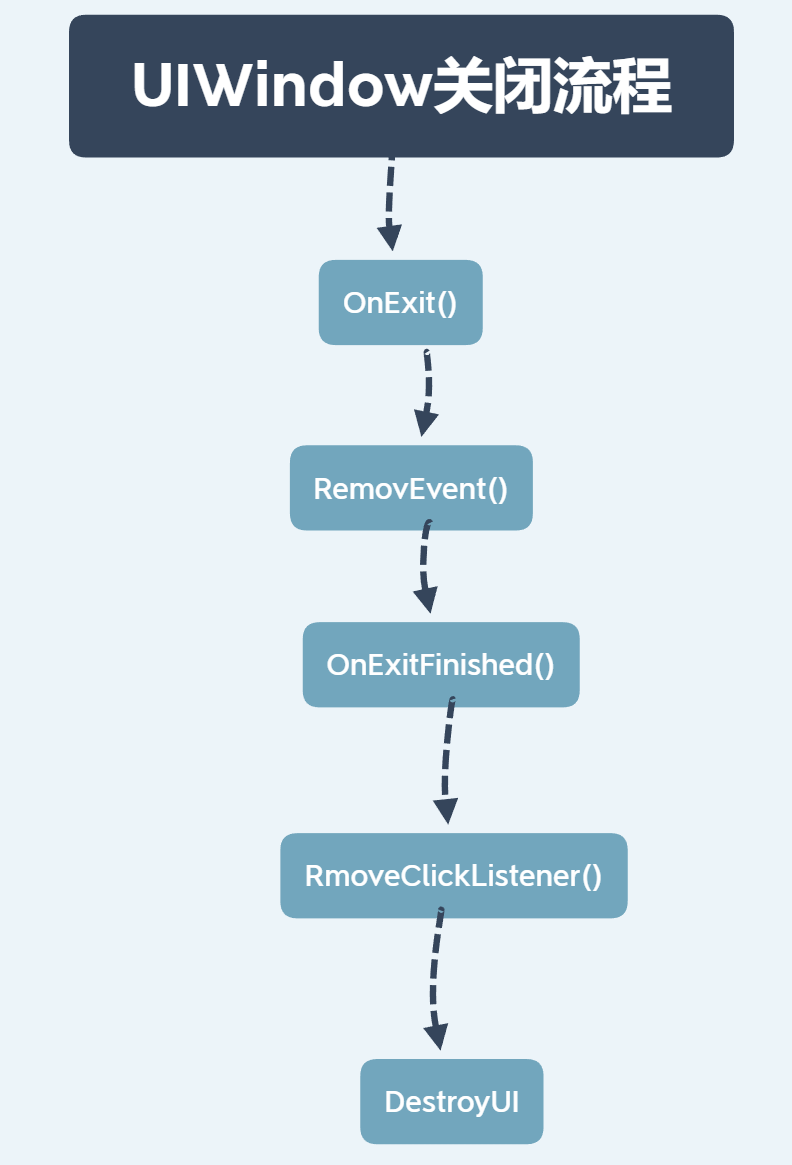
**（2）：AddClickListener**：BuildUI完成后，开始执行按钮，ScrollView等等UI组件的 监听绑定，

**（3）：OnEnter**：界面开始准备打开，在这里需要进行对界面的一些简单刷新操作，为后面动画的播放做准备。 比如刷新UI文字，UI图片。

**（4）：AddEvent**：界面开始绑定 游戏逻辑事件 监听，比如背包物品变化通知。

**（5）：OnEnterFinished**：界面已经完成进入，可以开始进行一些复杂操作了，比如打开背包时，格子的实例化刷新可以放在OnEnterFinished里面。

关闭流程如下：



**（1）：OnExit**：跟OnEnter对应，界面准备开始关闭前的处理函数。

**（2）：RemoveEvent**：跟AddEvent对应，移除游戏逻辑事件的监听。

**（3）：OnExitFinshed**：关闭动画播放完成，界面正式关闭，开始执行一些资源Release等操作。

**（4）：RemoveClickListener**：开始移除按钮等UI组件的监听事件，跟AddClickListener对应。

**（5）：DestroyUI**：界面开始要被销毁了，需要开始释放界面 所占用 的所有资源。

## 代码执行逻辑流程图：