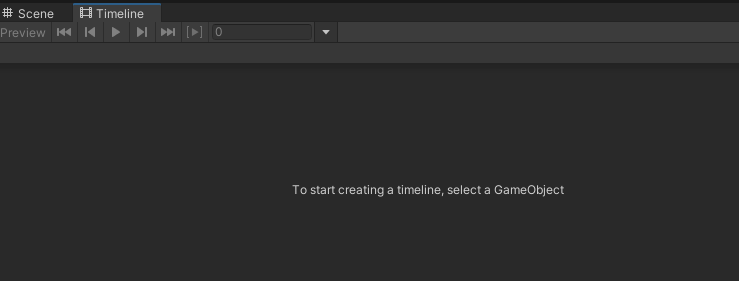
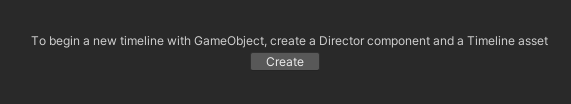
1. 创建Timeline：打开面板Window->Sequencing->Timeline



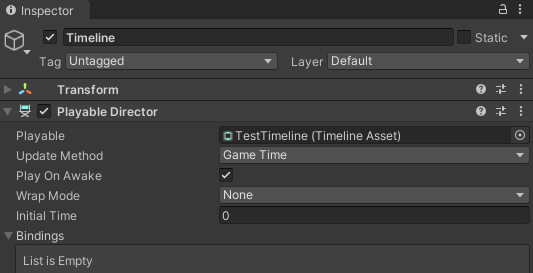
(1.1)选择一个要添加 Timeline 的物体，我创建一个物体就叫 Timeline(可以随意命名)，选择Timeline，然后在面板上显示 Create 按钮，如下



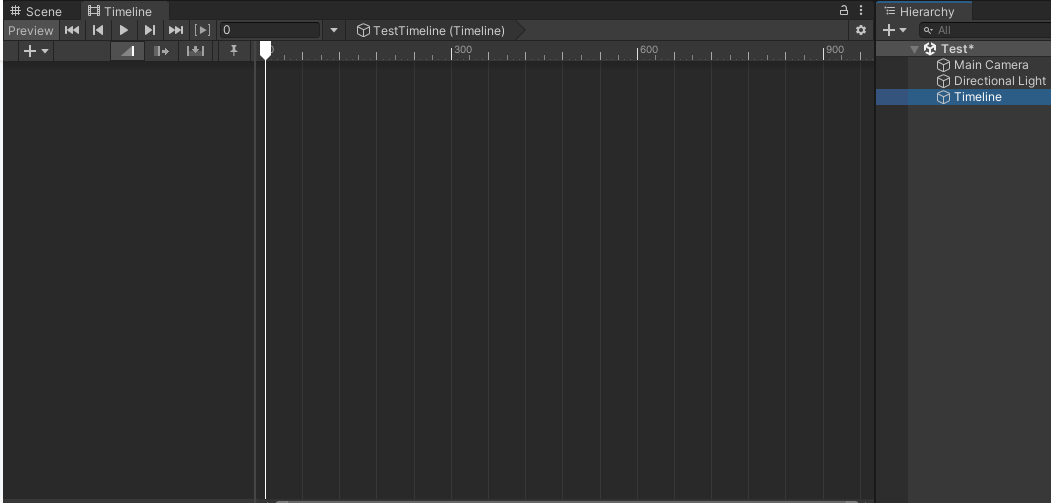
点 Create 按钮，保存一个 \*.playable 的文件,然后自动给Timeline物体添加一个 PlayableDirector 脚本

脚本参数 Playable ：TestTimeline 就是创建的 TestTimeline.playable 文件

Play On Awake：勾选则自动播放

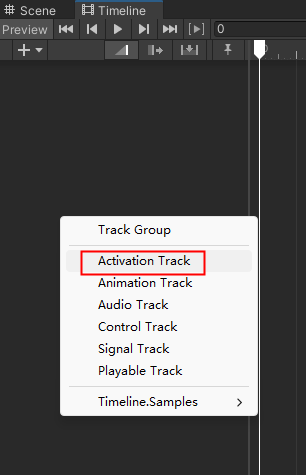


选择 Timeline 面板如下

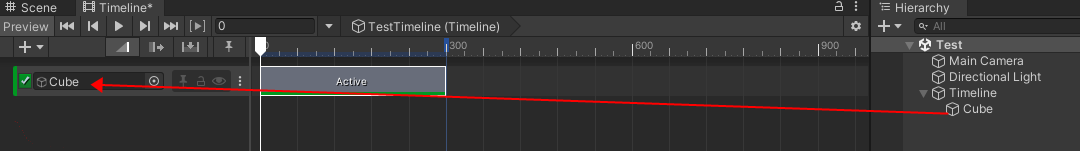


2.创建Activation Track (显示/隐藏物体轨道)

(2.1)在左侧空白处，鼠标右键，选择 ActivationTrack



(2.2) 创建一个 Cube，为了方便管理，放在 Timeline 下方

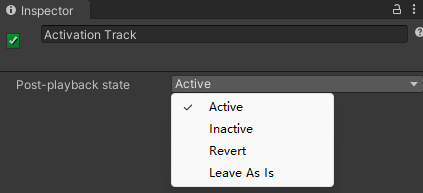


然后拖拽到 Activation 轨道的目标物体上

右侧Active区域是Cube物体显示的时间区域（Timeline执行到这个时间段，物体是显示的,在这个时间段之外时，物体都是隐藏的）

可以左右拖动，可以把鼠标放在左右两侧拉长时间区域

(2.3)点击左侧对勾区域，Post-playback state



Active状态：Timeline完成播放时，将绑定的游戏对象的状态设置为激活(显示)状态

Inactive状态：Timeline完成播放时，将绑定的游戏对象的状态设置为停用(隐藏)状态

Revert状态：Timeline开始播放之前，将绑定的游戏对象的状态设置为停用(隐藏)状态

Leave As Is状态：Timeline完成播放时，绑定的对象状态是显示的就设置为显示状态，Timeline完成播放时，绑定的对象状态是隐藏的，就设置为隐藏状态

名词解释：

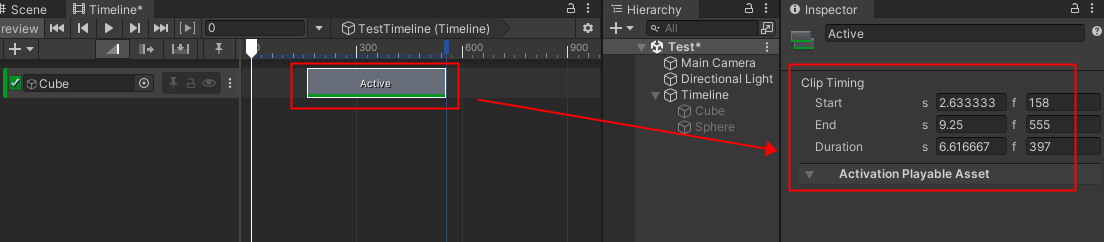
**Timeline完成播放时：**Timeline播放时间到，所有轨道结束时，才是**Timeline**完成播放时

(2.4)选择轨道上的 Active，在 Inspector 面板上可以看到参数

Start：开始时间(秒) f：帧(60帧/秒)

End：结束时间(秒)

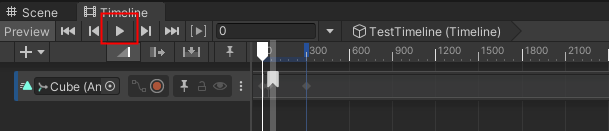
Duration：持续时间(秒)



预览：

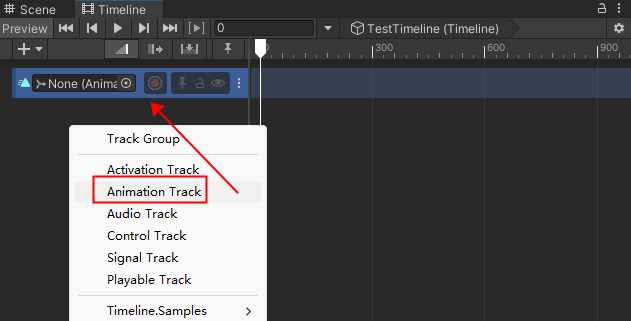
(2.4.1)选择挂Playable Director 脚本的GameObject，在 Timeline窗口左上角，点击三角预览播放

(2.4.2)直接运行Unity也可以预览播放

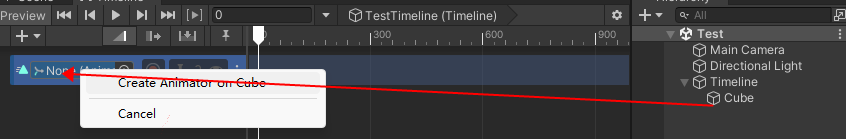


1. 创建 Animation Track (动画轨道)

(3.1)空白位置鼠标右键，选择 Animation Track



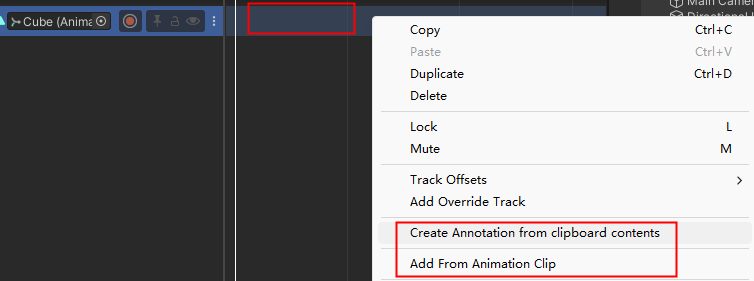
创建一个 Cube 拖拽到 None 位置



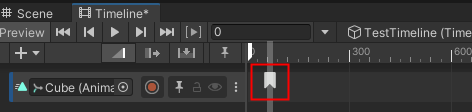
提示 Create Animator on Cube：添加Animator 组件到Cube上，点击Create 即可，自动在 Cube 上挂一个Animator 组件

(3.2)添加动画帧

在轨道空白位置鼠标右键



Create Annotation from clipboard contents：创建一个动画帧，在动画轨道上出现标签如下



AddFrom Animation Clip：弹出窗口选择一个现有的动画帧

也可以直接将一个动画帧拖拽到轨道上，如角色预制体拖拽到左侧作为绑定对象，然后将角色的动画拖拽到轨道上即可实现播放

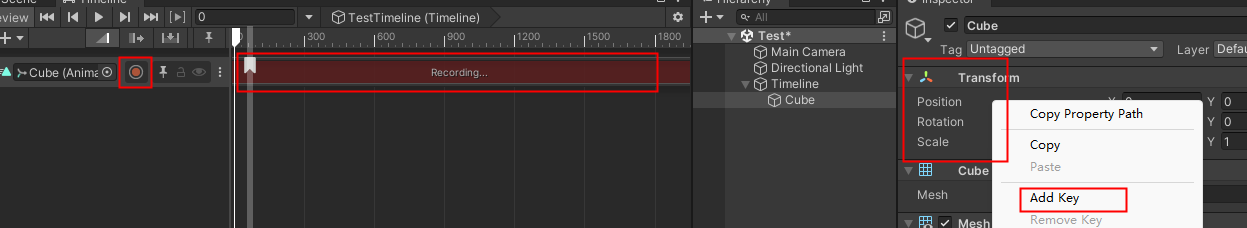


(3.3) 创建 Create Annotation from clipboard contents

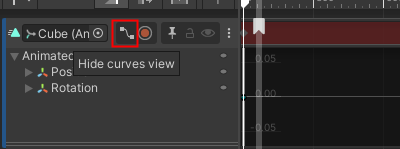
(3.3.1)点击左侧红色圆圈，开始录制，在轨道上显示红色并且有 Recoding... 字体提示

然后选择绑定的对象 Cube，在Inspector面板上，可以选择Cube上添加的可用组件添加动画帧

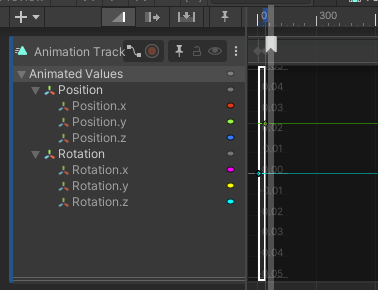
如Transform，可以分别在 Position、Rotation、Scale 位置鼠标右键 弹框选择 AddKey



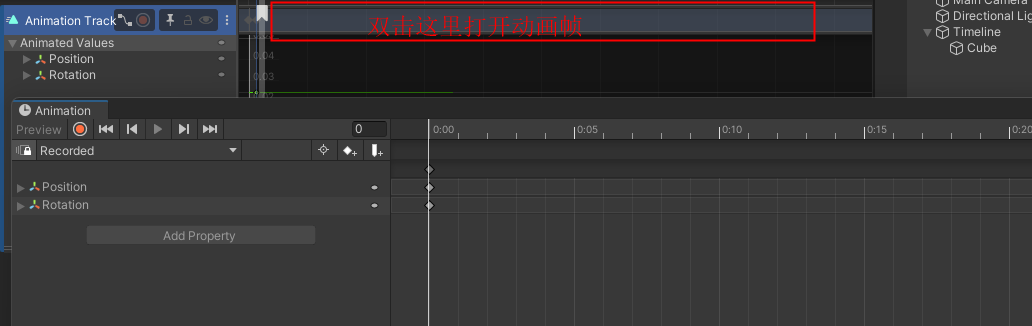
点击 AddKey将在左侧轨道下方出现选择的属性，点击下方图中红色位置，可以显示隐藏编辑属性



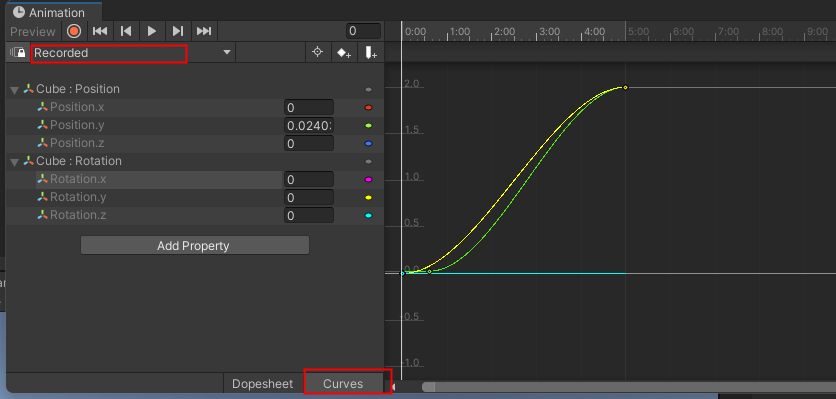
展开如下



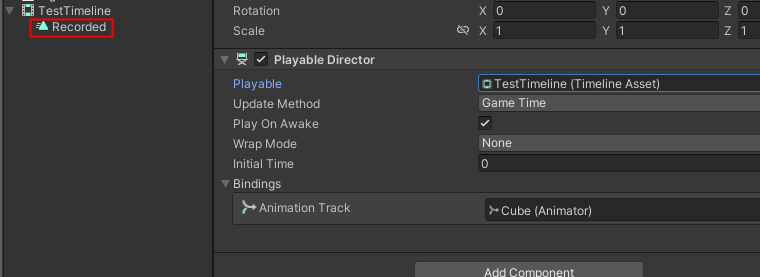
(3.3.2)在轨道区域双击打开动画编辑窗口



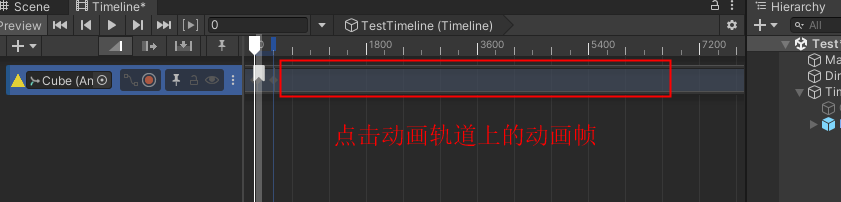
切换到 Curves 编辑动画

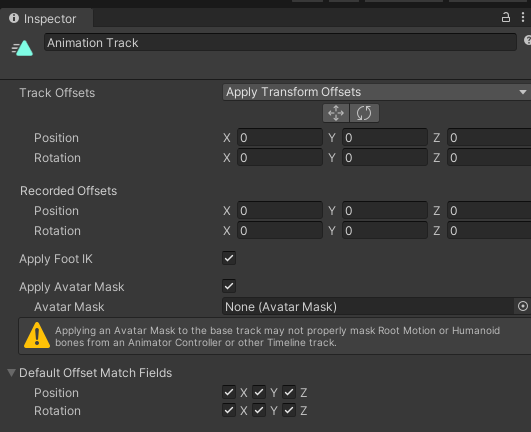


(3.3.3)左上角Recorded为保存的动画名，动画存放在当前编辑的TestTimeline.playable 下方，如下图，保存多个动画会自动命名为 Recorded(1)、Recorded(2)、Recorded(3)...



(3.3.4)点击动画轨道上的动画帧，在Inspector面板查看参数，如下





Track Offsets:将相同的位置和旋转偏移应用到动画轨道上所有的动画片段

Position: 添加控制Position 动画帧

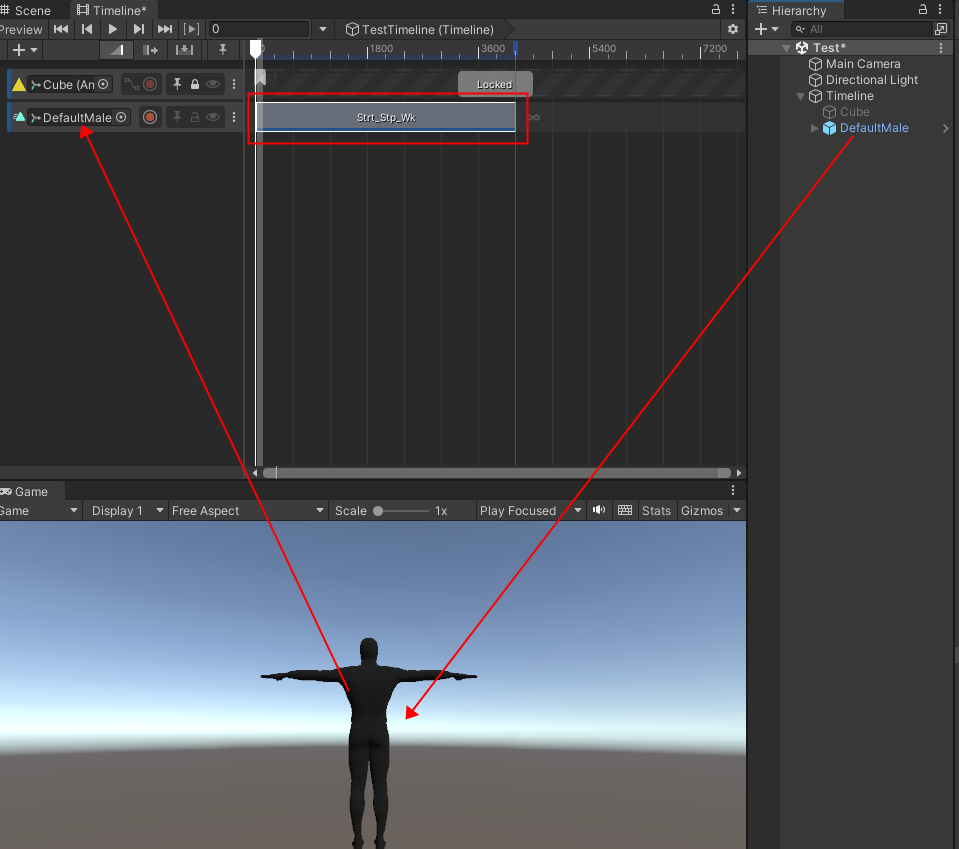
Rotation: 添加控制 Rotation 动画帧

Apply Foot IK：启用动画反向动力学功能

Apply Avatar Mask:启动/禁用动画骨骼遮罩，启动遮罩时，所选的遮罩会应用到当前动画轨道上所有的动画片段

Default Offset Match Fields：动画轨道上所有的动画片段在进行匹配片段偏移时，选择默认的匹配选项

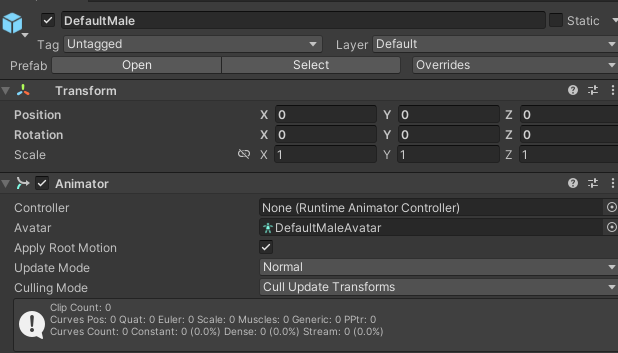
(3.4)AddFrom Animation Clip 添加一个现有动画帧



(3.4.1)将角色拖拽到左侧绑定  


模型需要添加Animator脚本

Animator属性Controller为空即可

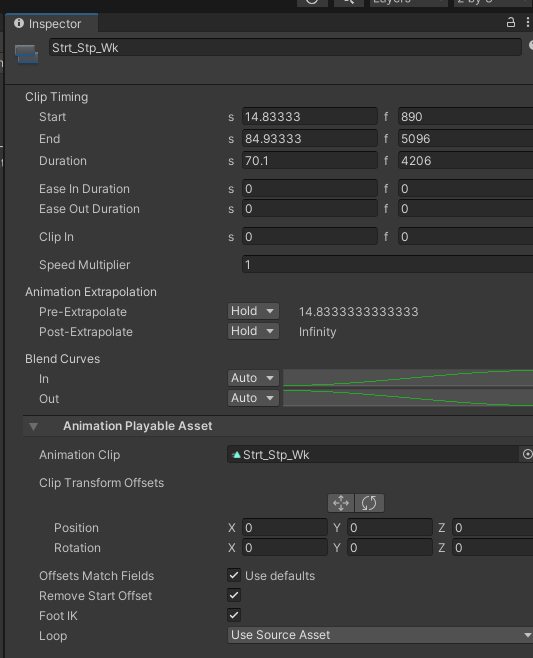


(3.4.2)在右侧轨道右键 AddFrom Animation Clip 或者拖拽一个动画帧到轨道上

运行或者预览即可查看动画播放

(3.4.3)在轨道上选择动画帧，在Inspector面板查看参数





可以修改片段的名称

(3.4.3.1)Clip Timing 属性包含一下内容

Start：动画片段开始时间

End：动画片段结束时间

Duration：动画片段持续时间

Ease In Duration：淡入动画片段所需时间

Ease Out Duration：淡出动画片段所需时间

(3.4.3.2)

Pre-Extrapolate:设置该动画帧开始播放前的状态，该属性会影响动画片段的淡入

None：在播放clip前Capsule会保持真实位置而不是动画位置

Hold：在播放clip前Capsule会保持第一帧动画位置

Loop：在播放clip前不停循环clip，保证在clip开始前动画内容正好播到结尾，不过开头不一定（主要看留出时间）

Ping Pong：会播放来回clip（会自动补动画），保证在clip开始时机前动画内容正好回到开头

Continue：只能在unity运行时查看，保持的是Loop或Hold选项的状态

Post-Extrapolate：设置动画片段的后外推，该属性影响动画片段的淡出

(3.4.3.3)Blend Curves:包含以下属性

In：自动/手动调整动画片段的淡入曲线

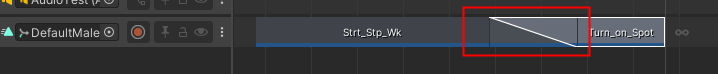
Out：自动/手动调整动画片段的淡出曲线

(3.4.3.4) Animation Playable Asset:包含以下属性

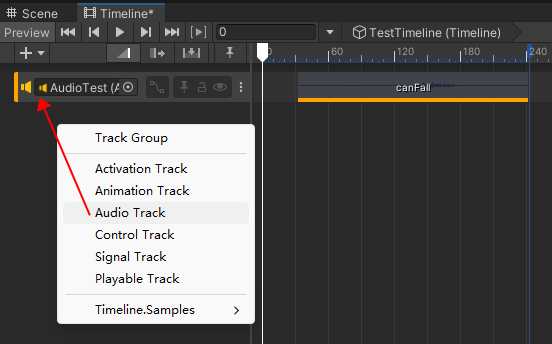
Clip Transform Offsets：将位置和旋转偏移应用于所选动画片段的根运动

Offset Match Fields：进行匹配片段偏移时，在片段级别上设置的匹配选项

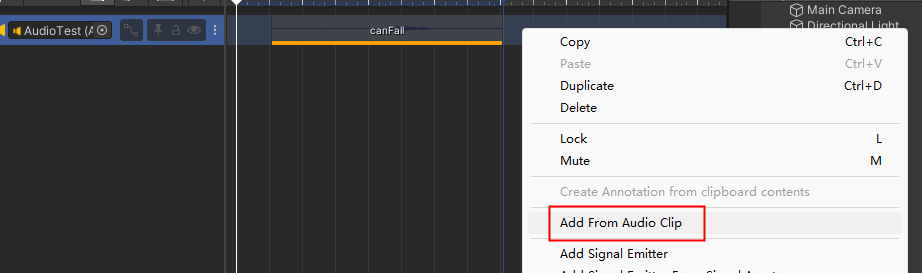
(3.4.4)动画融合，拖拽两个动画相互穿插控制融合时间



1. 创建 Audio Track (音效轨道)播放音效

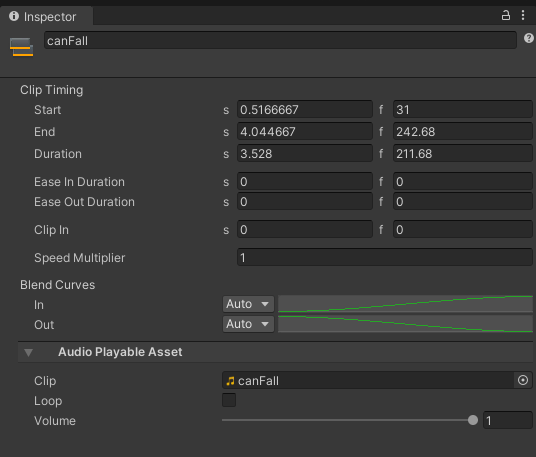


(4.1)轨道添加音效



轨道空白处右键 Add From Audio Clip 或者直接将音效拖拽到轨道上

(4.2)音效参数



点击轨道上音效片段，Inspector 面板查看参数

(4.2.1)Audio Playable Asset:包含以下属性

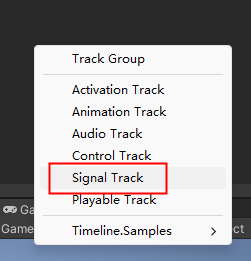
Clip：选择音频片段使用的音频文件。

Loop属性：设置音频片段是否循环播放。

Volume：设置音频片段的音量。

(5)Signal Track：信号轨道，发射信号，相当于发送一个事件

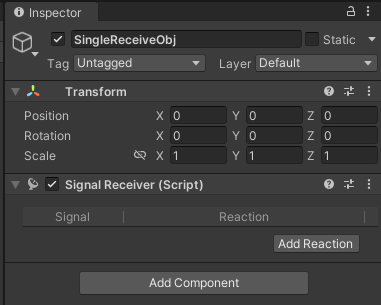
(5.1)在TimeWindow面板左侧空白区域，右键选择 Signal Track



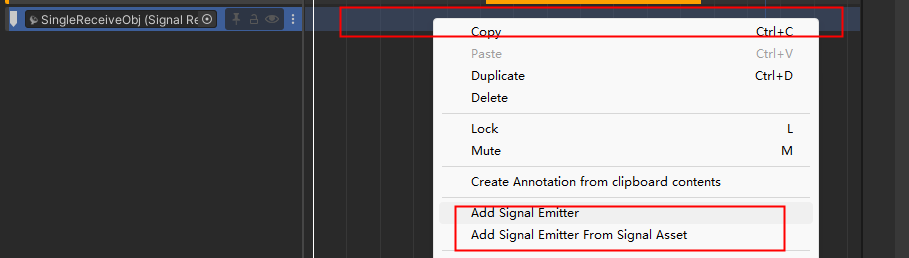


创建后显示如上，

(5.2)创建一个接收信号的 GameObject，我命名为 SingleReceiveObj，然后拖拽到左侧 None(Signal Receiver) 位置，提示添加 Create Signal Receiver 脚本，选择 Create 即可在 GameObject 上自动挂 Signal Receiver 脚本，如下



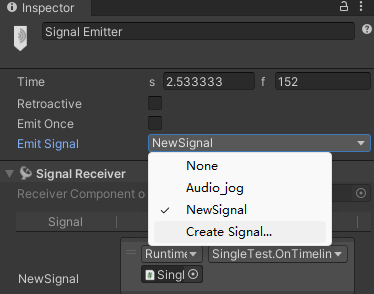
(5.3) 在信号轨道空白处右键添加一个发射器



Add Signal Emitter：创建一个信号发射器，发射器显示如下



点击信号发射器看Inspector 面板，Emit Signale 位置可以选择已经创建的发射器



Time：发射信号的时间

Retroactive：

EmitOnce：勾选后只会发射一次信号

Emit Signal：选择一个发射信号Asset，可以复用之前创建的，也可以选择 Create Signal 创建一个

Add Signal Emitter From Signal Asset: 会打开选择窗口，可以选择已经创建的发射器

(5.4)配置接收函数

// 创建一个接收信号的脚本，类名随意

public class SignalReceiveTest : MonoBehaviour

{

public void OnTimelineSignal()

{

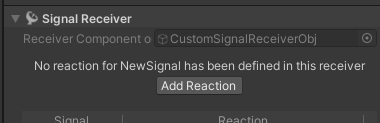
Debug.LogError("收到信号了");

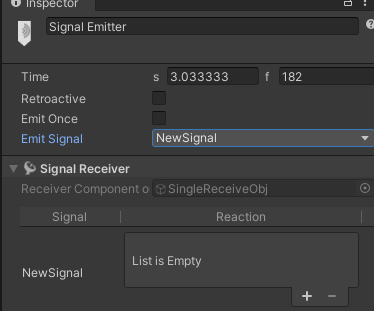
}

}

将 SignalReceiveTest 脚本挂到 SingleReceiveObj 物体上

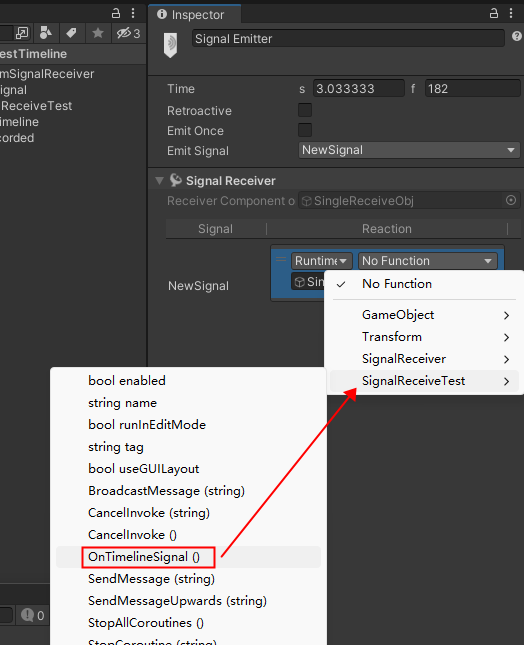
在 Signal Receiver 下方点击 AddReaction





点击 +

选择 Runtime，在 NoFunction 处下拉选择 SignalReceiveTest 脚本上的 OnTimelineSignal 函数



运行预览，时间轴执行到信号发射器位置时，将会调用到 OnTimelineSignal 函数

(5.5).自定义信号

(5.5.1)新建信号发射类，继承 SignalEmitter

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Timeline;

// 自定义的信号

public class CustomSignal : SignalEmitter

{

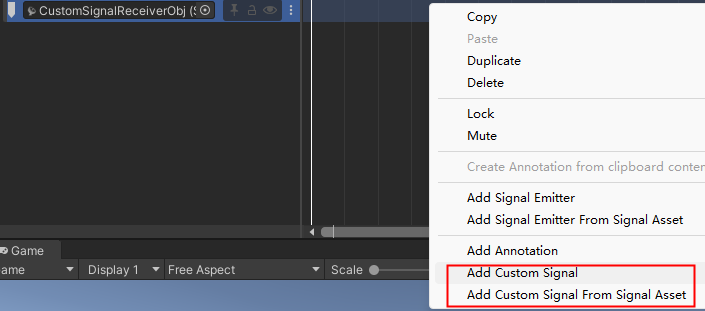
// 自定义参数

public string eventName;

public int param;

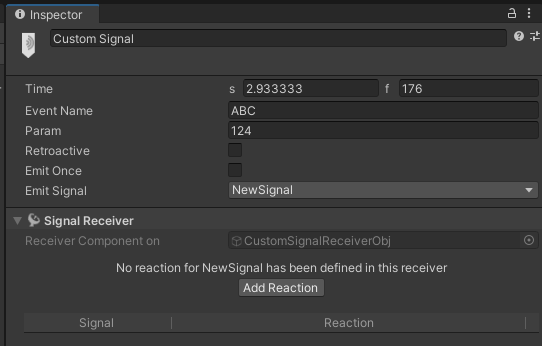
}

创建一个 Signal Track 轨道，然后在右侧右键，如下可以看到新建的信号发射器



创建一个接收信号的GameObject我命名为 CustomSignalReceiverObj，添加 SngnalReceiver 脚本，并拖拽到信号轨道

(5.5.2)选择新创建的 Custom Signal，在Inspector 面板可以看到 CustomSignal 类的两个参数 eventName和 param，可以手动输入参数值



(5.5.3) 创建信号接收类

/// <summary>

/// 自定义信号接收器

/// </summary>

public class CustomSignalReceiver : MonoBehaviour, INotificationReceiver

{

public void OnNotify(Playable origin, INotification notification, object context)

{

var signal = notification as CustomSignal;

if (signal != null

&& signal.asset != null)

{

Debug.LogError(signal.number + " " + signal.param);

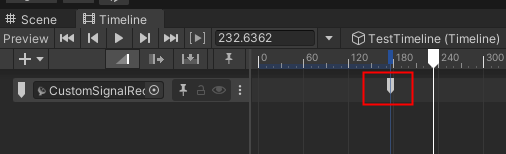
}

}

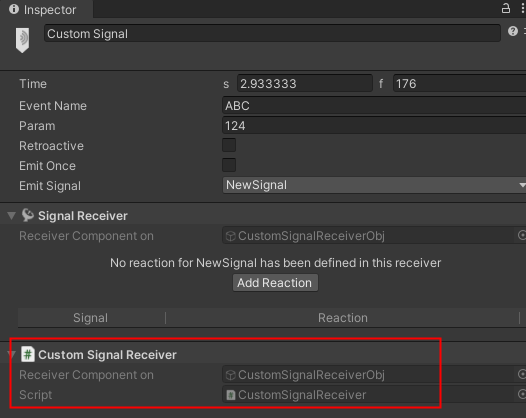
}

将脚本CustomSignalReceiver挂到 CustomSignalReceiverObj 上

选择添加的自定义信号发射器Custom Signal 如下

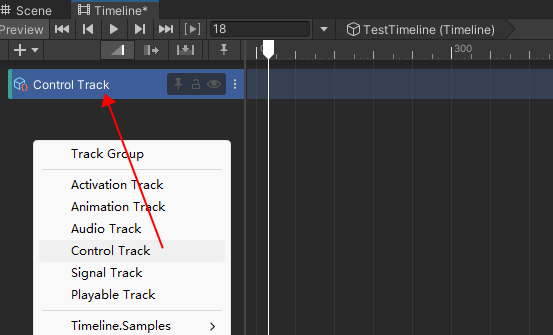


查看Inspector面板下方显示



运行后将自动触发 CustomSignalReceiver 类的 OnNotify 函数

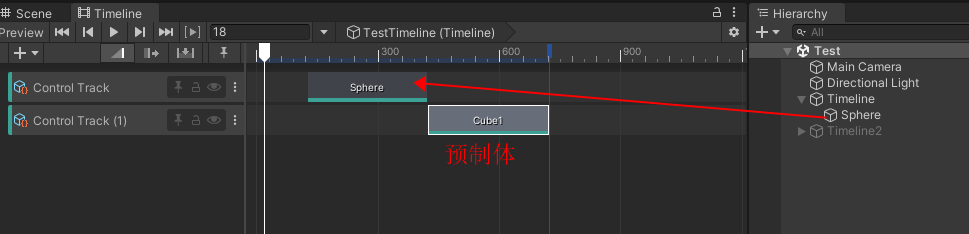
(6)创建 Control Track



(6.1)控制物体显示

方法一：将场景内的物体拖拽到右侧轨道上绑定，下方 Sphere

方法二：将预制体拖拽到右侧轨道上绑定，下方 Cube1

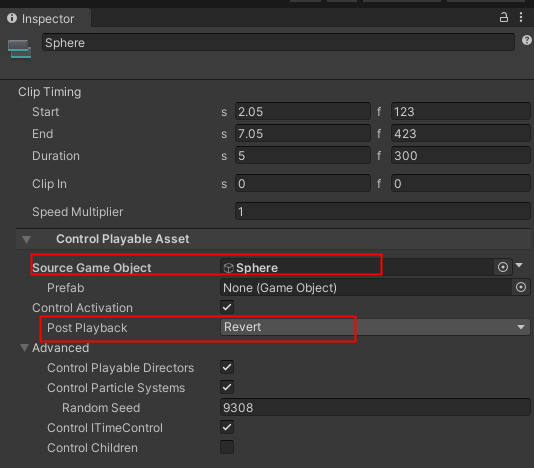


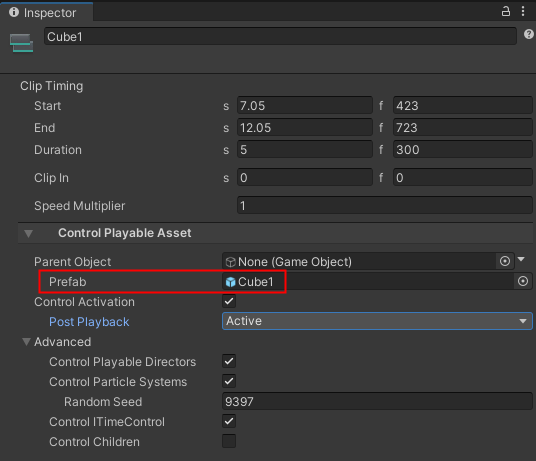
运行，时间轴执行到轨道开始时间时，将绑定物体显示出来，轨道时间结束后隐藏

Control Track 绑定的如果是场景内的物体，则直接控制其显示/隐藏

Control Track 绑定的如果是预制体，可以自动加载预制体并控制其显示隐藏

(6.2)点击轨道上的片段，在Inspector 面板查看参数





SourceGameObject：绑定的物体是场景内的GameObject 如上 Sphere

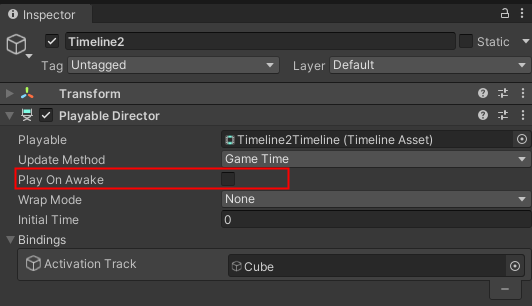
Prefab：预制体拖拽到轨道上，Prefab属性关联对绑定对象的引用，如上 Cube1

Post PlayBack 同 (2.3) Post-playback state

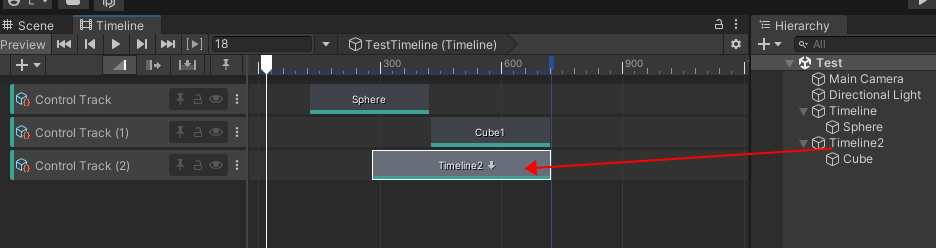
(6.3)Control Track嵌套控制其他的Timeline

再创建一个Timeline2

Playable Director 脚本Play On Awake 不勾选(不自动播放)



在 Timeline上创建一个Control Track，将Timeline2 拖拽到右侧轨道上，嵌套到Timeline

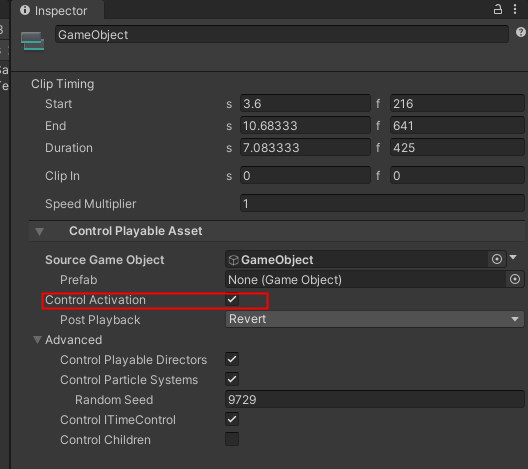


(6.4)在 Timeline选择被嵌套的 Timeline2，查看Inspector面板

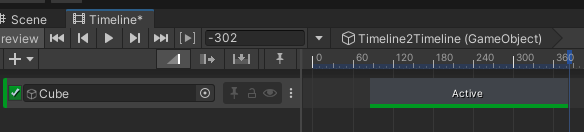
Control Activation：

勾选时，Timeline 执行时会调用 Timeline2，主要让Timeline2做一些前置处理

不勾选时，Timeline执行时需要等到Timeline2时间开始位置才触发Timeline2



看例子，下面是 Timeline2控制一个Cube的显示

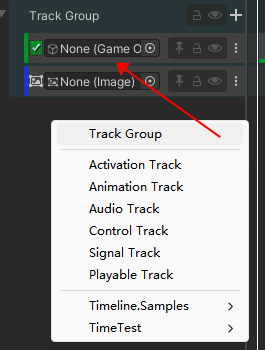


(6.4.1)如果在Timeline轨道上将Timeline2 的 Control Activation **勾选**，则Timeline一开始执行，Timeline2控制Cube 立即隐藏，直到执行到 Timeline2需要显示Cube的时候，才将 Cube显示出来

(6.4.2)如果在Timeline轨道上将Timeline2 的 Control Activation **不勾选**，则Timeline一开始执行，Timeline2 不会控制Cube隐藏(Cube一开始是显示的)，知道执行到Timeline2开始时间位置，Timeline2才控制Cube先隐藏，然后时间到 Active 位置时将Cube显示

1. 添加 TrackGroup(轨道组)

将多个轨道规划为一个组，方便管理

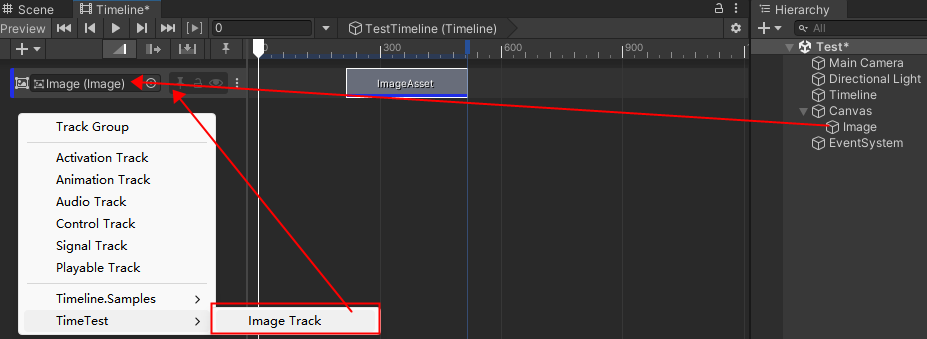


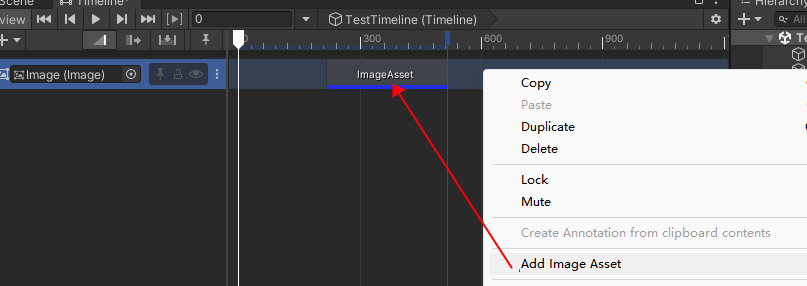
(8)添加自定义轨道

(8.1)我添加一个设置 Image 颜色的 自定义轨道，执行到轨道帧开始位置时改变Image颜色，轨道帧结束时将Image颜色设置为(0, 0, 0,0)

先看效果，下图中 TimeTest->Image Track 是自定义的轨道

绑定的对象类型为 UGUI->Image





在轨道右侧，鼠标右键添加轨道帧，Add Image Asset 也是自定义添加的

(8.2)选中 ImageAsset 在 Inspector 面板查看参数

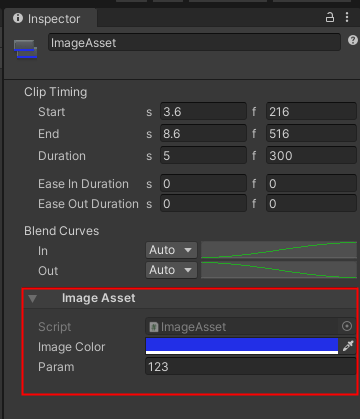


Image Asset 部分

Image Color：添加的自定义颜色属性，执行时将Image 的颜色设置为这个颜色

Param：添加的自定义 Int 类型属性

需要添加三个脚本

(8.3) 添加轨道帧资源，上方右键Add Image Asset 就是下面代码创建的

using System;

using UnityEngine;

using UnityEngine.Playables;

namespace TimeTest

{

/// <summary>

/// 创建轨道资源

/// </summary>

[Serializable]

public class ImageAsset : PlayableAsset

{

// 设置的颜色

public Color imageColor;

// 参数

public int param;

public override Playable CreatePlayable(PlayableGraph graph, GameObject owner)

{

// 创建一个新的 Playable(Script类型)

// ScriptPlayable<ImageMixerBehavior>.Create 实际接收两个参数

// 第一个参数是 Graph

// 第二个参数是 我们创建的这个Playable接收几个参数，默认不填写那么就是0个输入

var playable = ScriptPlayable<ImageMixerBehavior>.Create(graph);

// 通过 GetBehaviour 获取上面创建的 ImageMixerBehavior 类型实例

var imageMixerBehavior = playable.GetBehaviour();

// 将轨道资源参数赋值给 imageMixerBehavior

imageMixerBehavior.imageColor = imageColor;

imageMixerBehavior.param = param;

// 返回 Playable 类型实例，Unity会帮我们自动连接

return playable;

}

}

}

(8.4)添加轨道，TimeTest->Image Track 就是下面代码创建的

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Timeline;

using UnityEngine.UI;

namespace TimeTest

{

/// <summary>

/// 自定义 Timeline 轨道 ImageTrack （名字随意定）

/// 在 Timeline 添加轨道位置右键，新加 TimeTest->ImageTrack

/// </summary>

[TrackColor(0.13f, 0.18f, 0.9f)] // 轨道颜色

[TrackBindingType(typeof(Image))] // 绑定对象类型为 UnityEngine.UI.Image

[TrackClipType(typeof(ImageAsset))] // 轨道帧类型为 ImageAsset (也需要自定义)

public class ImageTrack : TrackAsset

{

public override Playable CreateTrackMixer(PlayableGraph graph, GameObject go, int inputCount)

{

return ScriptPlayable<ImageMixerBehavior>.Create(graph, inputCount);

}

}

}

(8.5)添加轨道执行逻辑

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.UI;

namespace TimeTest

{

/// <summary>

/// 继承自 PlayableBehaviour，定义 Playable 的行为

/// </summary>

public class ImageMixerBehavior : PlayableBehaviour

{

private Image img;

public Color imageColor;

public int param;

/// <summary>

/// 重写 OnBehaviourPlay 函数，第一次执行到轨道帧开始的时间

/// </summary>

/// <param name="playable"></param>

/// <param name="info"></param>

public override void OnBehaviourPlay(Playable playable, FrameData info)

{

base.OnBehaviourPlay(playable, info);

}

/// <summary>

/// 重写 ProcessFrame 函数，Timeline 开始执行，直到所有轨道结束，每帧都会调用这个方法

/// </summary>

/// <param name="playable"></param>

/// <param name="info"></param>

/// <param name="playerData">绑定对象 类型为：ImageTrack 类设置的 TrackBindingType 类型 Image</param>

public override void ProcessFrame(Playable playable, FrameData info, object playerData)

{

Color blendColor = Color.clear;

// 转换为绑定对象 Image

img = playerData as Image;

int inputCount = playable.GetInputCount();

if (null != img && inputCount > 0)

{

for (int i = 0; i < inputCount; i++)

{

float weight = playable.GetInputWeight(i);

ImageMixerBehavior imageMixerBehavior = ((ScriptPlayable<ImageMixerBehavior>)playable.GetInput(i)).GetBehaviour();

// 获取颜色值

blendColor += imageMixerBehavior.imageColor \* weight;

}

// 给绑定的 Image 对象设置颜色

img.color = blendColor;

}

}

/// <summary>

/// 执行到当前轨道帧 End Time

/// </summary>

/// <param name="playable"></param>

/// <param name="info"></param>

public override void OnBehaviourPause(Playable playable, FrameData info)

{

base.OnBehaviourPause(playable, info);

if (null != img)

{

img.color = Color.clear;

}

}

}

}

执行即可预览效果