# Playables API

Playables API 提供一种通过组织和评估树状结构（称为 PlayableGraph）中的数据源来创建工具、效果或其他游戏机制的方法。PlayableGraph 允许您混合、融合和修改多个数据源，并通过单个输出播放它们。

Playables API 支持动画、音频和脚本。Playables API 还提供通过脚本与[动画系统](file:///E:\UnityDocumentation\Manual\AnimationSection.html)和音频系统进行交互的能力。

尽管 Playables API 目前仅限于动画、音频和脚本，但它是一种通用 API，最终可供视频和其他系统使用。

## 可播放项 (Playable) 与动画组件

动画系统已有一个图形编辑工具，这是一个仅限于播放动画的状态机系统。Playables API 设计得更灵活并支持其他系统。Playables API 还可创建状态机无法实现的图形。这些图形表示一个数据流，指示每个节点生成和使用的内容。此外，单个图形不限于单个系统。单个图形可能包含动画、音频和脚本的节点。

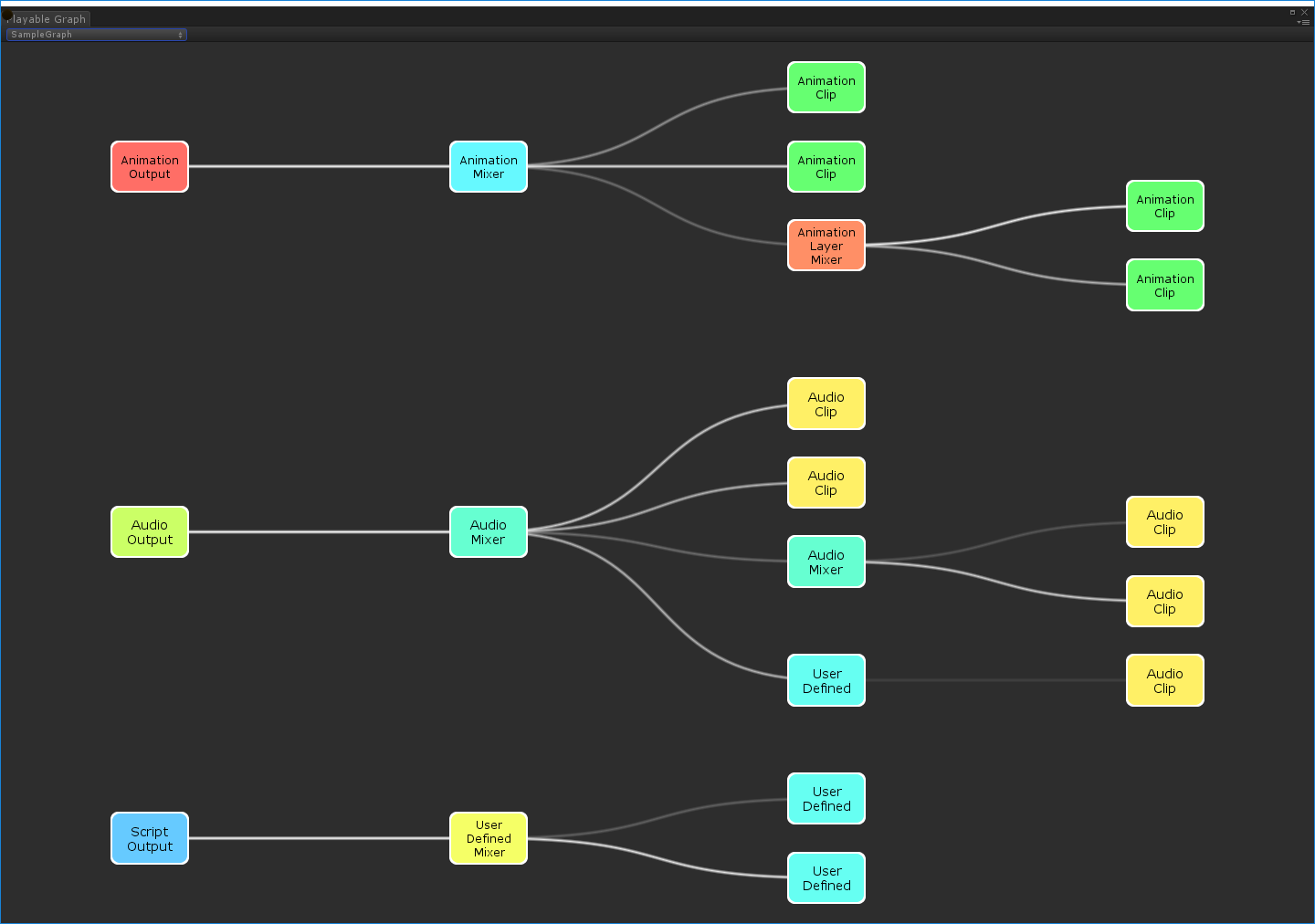
## Playables API 的优点

* Playables API 支持动态动画混合。这意味着场景中的对象可以提供自己的动画。例如，武器、宝箱和陷阱的动画可以动态添加到 PlayableGraph 并使用一段时间。
* Playables API 可让您轻松播放单个动画，而不会产生创建和管理 AnimatorController 资源所涉及的开销。
* Playables API 允许用户动态创建混合图并直接逐帧控制混合权重。
* 可在运行时创建 PlayableGraph，根据条件按需添加可播放节点。可量身定制 PlayableGraph 来适应当前情况的要求，而不是提供一个巨大的“一刀切”图形来启用和禁用节点。

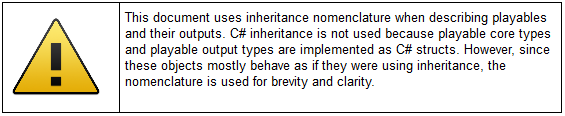
# PlayableGraph

PlayableGraph 定义一组绑定到 GameObject 或[组件](file:///E:\\UnityDocumentation\\Manual\\Components.html)的可播放输出项。PlayableGraph 还定义一组可播放项及其依赖关系。图 1 提供了一个示例。

PlayableGraph 负责管理可播放项及其输出的生命周期。使用 PlayableGraph 可创建、连接和销毁可播放项。

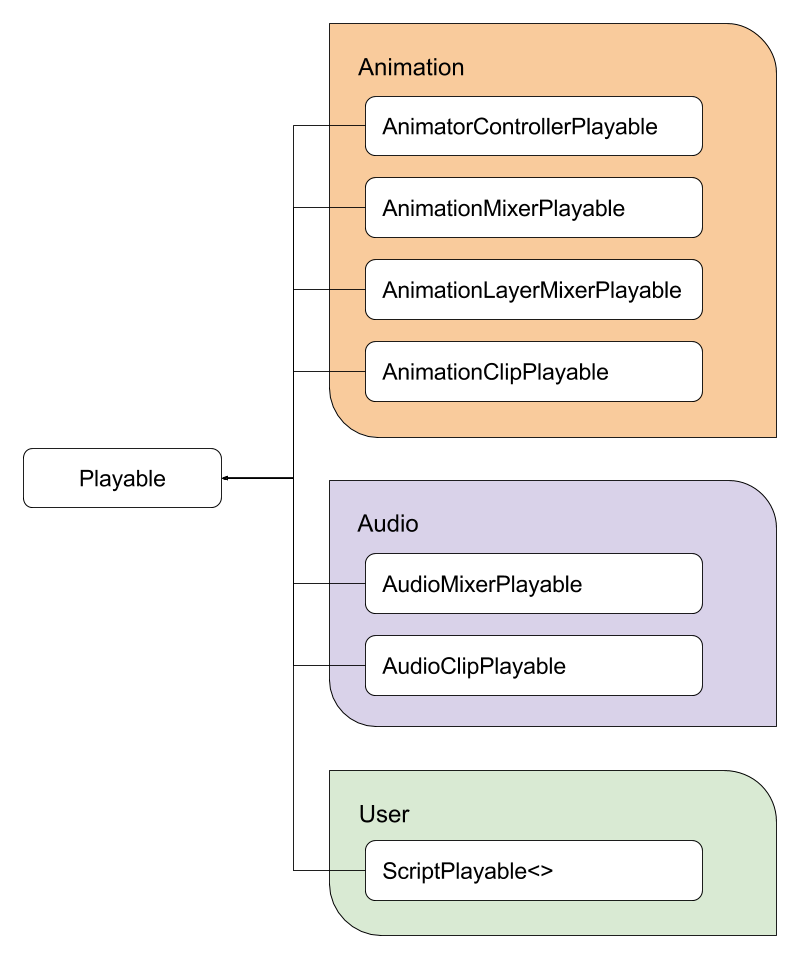
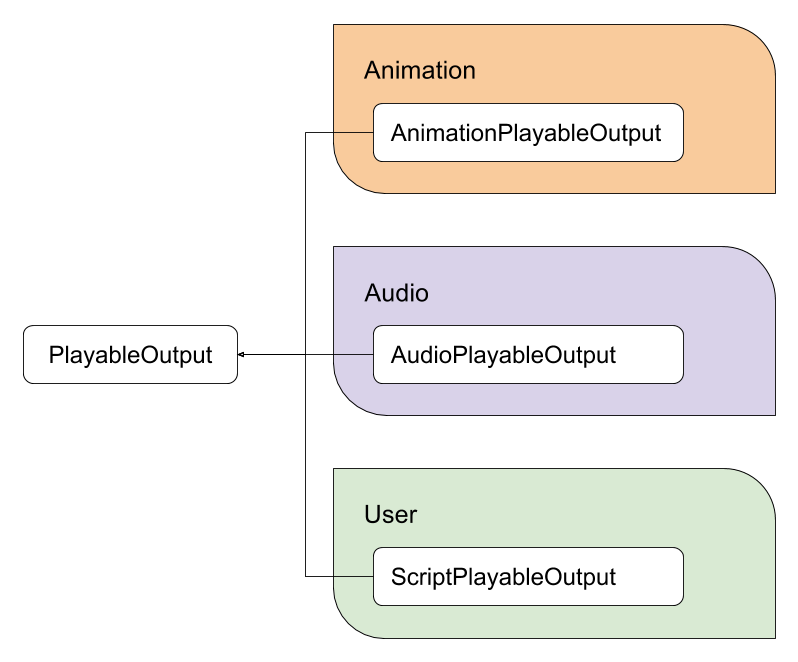
图 1：PlayableGraph 示例

在图 1 中，显示 PlayableGraph 时，为了紧凑，从图形节点的名称中移除了“Playable”一词。例如，名为“AnimationClipPlayable”的节点显示为“AnimationClip”。



可播放项是一种继承 IPlayable 接口的 C# 结构体。它用于定义其与其他可播放项的关系。同样，可播放项输出是继承 IPlayableOutput 接口的 C# 结构体，用于定义 PlayableGraph 的输出。

图 2 显示了最常见的核心可播放项类型。图 3 显示了核心可播放项输出类型。

图 2：核心可播放项类型图 3：核心可播放项输出类型

可播放项核心类型和可播放项输出类型以 C# 结构的形式实现，从而避免为[收集垃圾而分配内存](file:///E:\UnityDocumentation\Manual\UnderstandingAutomaticMemoryManagement.html)。

“Playable”是所有可播放项的基本类型，这意味着您总是可以隐式将一个可播放项转换为 Playable。相反的情况并不成立，如果将“Playable”显式转换为不兼容类型，则会抛出异常。它还定义能在可播放项上执行的所有基本方法。要访问特定于类型的方法，您需要将可播放项转换为相应类型。

PlayableOutput 也是如此，它是所有可播放项输出的基本类型，定义了基本方法。

注意：Playable 和 PlayableOutput 未暴露大量方法。但“PlayableExtensions”和“PlayableOutputExtensions”静态类提供了扩展方法。

所有非抽象可播放项都有一个公有静态方法 Create()，该方法创建相应类型的可播放项。“Create()”方法始终将 PlayableGraph 作为其第一个参数，该图拥有新创建的可播放项。某些类型的可播放项可能需要其他参数。非抽象可播放项输出还会暴露 Create() 方法。

有效的可播放项输出应链接到可播放项。如果可播放项输出未链接到可播放项，则可播放项输出不执行任何操作。要将可播放项输出链接到可播放项，请使用 PlayableOutput.SetSourcePlayable() 方法。对于该特定可播放项输出，链接的可播放项充当可播放项树的根。

要将两个可播放项连接在一起，请使用 PlayableGraph.Connect() 方法。请注意，某些可播放项不能有输入。

使用 PlayableGraph.Create() 静态方法来创建 PlayableGraph。

使用 PlayableGraph.Play() 方法来播放 PlayableGraph。

使用 PlayableGraph.Stop() 方法来停止播放 PlayableGraph。

使用 PlayableGraph.Evaluate() 方法来评估 PlayableGraph 在特定时间的状态。

使用 PlayableGraph.Destroy() 方法来手动销毁 PlayableGraph。此方法会自动销毁 PlayableGraph 创建的所有可播放项和可播放项输出。必须手动调用此销毁方法来销毁 PlayableGraph，否则 Unity 会发出一条错误消息。

# ScriptPlayable 和 PlayableBehaviour

要创建自定义的可播放项，必须从 PlayableBehaviour 基类继承。 public class MyCustomPlayableBehaviour : PlayableBehaviour { // 自定义可播放项行为的实现 // 根据需要重载 PlayableBehaviour 方法 }

要将 PlayableBehaviour 用作自定义可播放项，还必须将其封装在 ScriptPlayable<> 对象内。如果不具备自定义可播放项的实例，可通过调用以下函数为对象创建 ScriptPlayable<>：

ScriptPlayable<MyCustomPlayableBehaviour>.Create(playableGraph);

如果已有自定义可播放项的实例，可通过调用以下函数用 ScriptPlayable<> 来包裹该实例：

MyCustomPlayableBehaviour myPlayable = new MyCustomPlayableBehaviour();

ScriptPlayable<MyCustomPlayableBehaviour>.Create(playableGraph, myPlayable);

此情况中将克隆该实例，然后将实例分配给 ScriptPlayable<>。实际上，此代码与先前代码执行的操作完全相同；不同之处在于 myPlayable 可能是将要在 Inspector 中配置的公有属性，然后可为脚本的每个实例设置行为。

您可以使用 ScriptPlayable<T> .GetBehaviour() 方法从 ScriptPlayable<> 获取 PlayableBehaviour 对象。

# 可播放项 (Playable) 示例

## PlayableGraph Visualizer

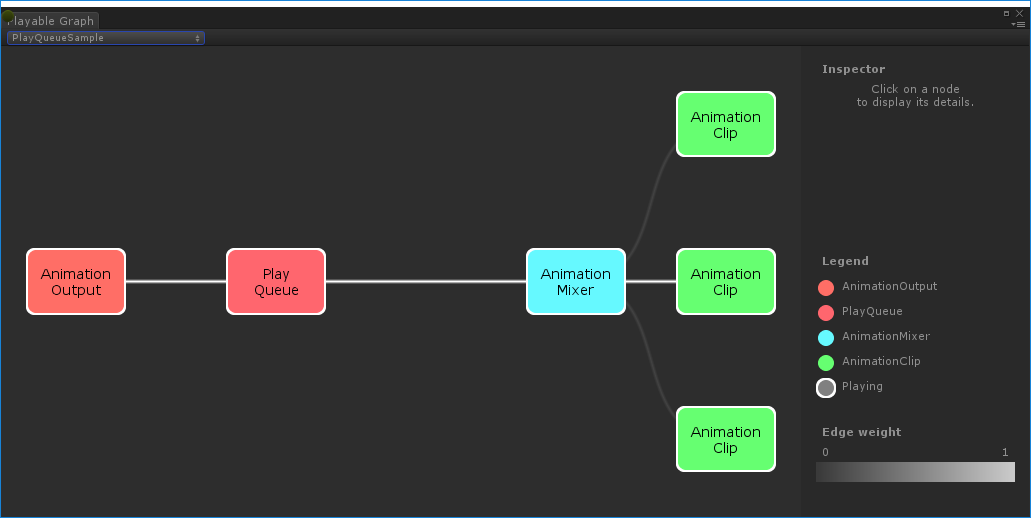
本文档中的所有示例都使用 PlayableGraph Visualizer（如下图所示）来说明 Playables API 创建的树和节点。PlayableGraph Visualizer 是通过 GitHub 提供的工具。

要使用 PlayableGraph Visualizer，请执行以下操作：

1.从 [GitHub 代码仓库](https://github.com/UnityTech/graph-visualizer)下载与您的 Unity 版本对应的 PlayableGraph Visualizer

2.通过选择 **Window** > **PlayableGraph Visualizer** 打开该工具

3.使用 GraphVisualizerClient.Show(PlayableGraph graph, string name) 来注册您的图。

GraphVisualizer 窗口

图中的可播放项 (Playable) 以彩色节点表示。线条颜色强度表示混合的权重。请参阅 [GitHub](https://github.com/UnityTech/graph-visualizer) 以了解关于此工具的更多信息。

## 播放游戏对象上的单个动画剪辑

以下示例演示一个简单的 PlayableGraph，它有一个链接到单个可播放节点的可播放项输出。可播放节点将播放单个动画剪辑（剪辑）。AnimationClipPlayable 必须包裹动画剪辑，使其与 Playables API 兼容。

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class PlayAnimationSample : MonoBehaviour

{

public AnimationClip clip;

PlayableGraph playableGraph;

void Start()

{

playableGraph = PlayableGraph.Create();

playableGraph.SetTimeUpdateMode(DirectorUpdateMode.GameTime);

var playableOutput = AnimationPlayableOutput.Create(playableGraph, "Animation", GetComponent<Animator>());

// 将剪辑包裹在可播放项中

var clipPlayable = AnimationClipPlayable.Create(playableGraph, clip);

// 将可播放项连接到输出

playableOutput.SetSourcePlayable(clipPlayable);

// 播放该图。

playableGraph.Play();

}

void OnDisable()

{

//销毁该图创建的所有可播放项和 PlayableOutput。

playableGraph.Destroy();

}

}

PlayAnimationSample 生成的 PlayableGraph

使用 AnimationPlayableUtilities 来简化动画可播放项的创建和播放，如以下示例中所示：\_\_ \_\_

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class PlayAnimationUtilitiesSample : MonoBehaviour

{

public AnimationClip clip;

PlayableGraph playableGraph;

void Start()

{

AnimationPlayableUtilities.PlayClip(GetComponent<Animator>(), clip, out playableGraph);

}

void OnDisable()

{

// 销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

## 创建动画混合树

以下示例演示如何使用 AnimationMixerPlayable 来混合两个动画剪辑。在混合动画剪辑之前，它们必须由可播放项包裹。为此，AnimationClipPlayable（clipPlayable0 和 clipPlayable1）包裹每个 AnimationClip（clip0 和 clip1）。SetInputWeight() 方法动态调整每个可播放项的混合权重。

虽然在此示例中未显示 AnimationMixerPlayable，但也可使用它来混合可播放的混合器和其他可播放项。

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class MixAnimationSample : MonoBehaviour

{

public AnimationClip clip0;

public AnimationClip clip1;

public float weight;

PlayableGraph playableGraph;

AnimationMixerPlayable mixerPlayable;

void Start()

{

// 创建该图和混合器，然后将它们绑定到 Animator。

playableGraph = PlayableGraph.Create();

var playableOutput = AnimationPlayableOutput.Create(playableGraph,"Animation", GetComponent<Animator>());

mixerPlayable = AnimationMixerPlayable.Create(playableGraph, 2);

playableOutput.SetSourcePlayable(mixerPlayable);

// 创建 AnimationClipPlayable 并将它们连接到混合器。

var clipPlayable0 = AnimationClipPlayable.Create(playableGraph, clip0);

var clipPlayable1 = AnimationClipPlayable.Create(playableGraph, clip1);

playableGraph.Connect(clipPlayable0, 0, mixerPlayable, 0);

playableGraph.Connect(clipPlayable1, 0, mixerPlayable, 1);

//播放该图。

playableGraph.Play();

}

void Update()

{

weight = Mathf.Clamp01(weight);

mixerPlayable.SetInputWeight(0, 1.0f-weight);

mixerPlayable.SetInputWeight(1, weight);

}

void OnDisable()

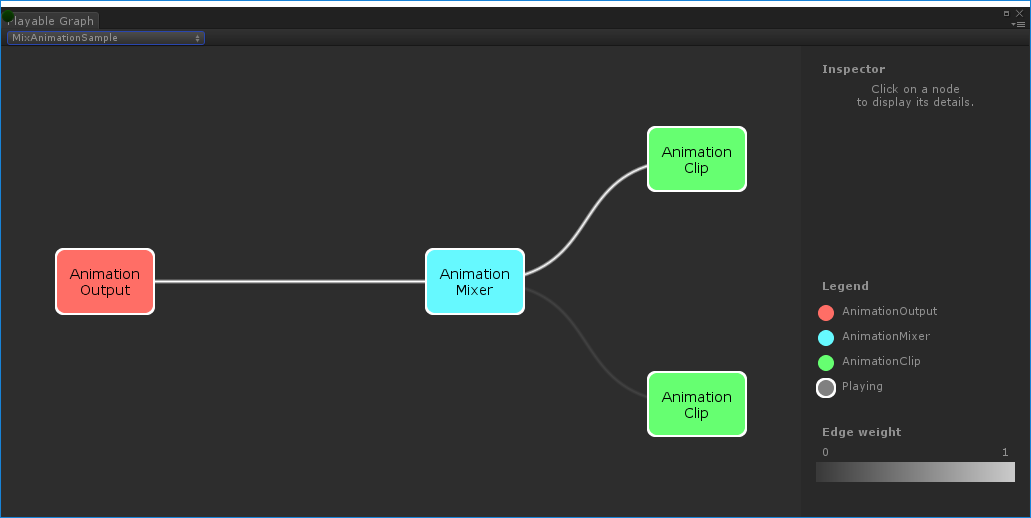
{

//销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

MixAnimationSample 生成的 PlayableGraph

## 混合 AnimationClip 和 AnimatorController

以下示例演示如何使用 AnimationMixerPlayable 来混合 AnimationClip 与 AnimatorController。

在混合 AnimationClip 和 AnimatorController 之前，它们必须由可播放项包裹。为此，AnimationClipPlayable (clipPlayable) 包裹 AnimationClip (clip)，而 AnimatorControllerPlayable (ctrlPlayable) 包裹 RuntimeAnimatorController (controller)。SetInputWeight() 方法动态调整每个可播放项的混合权重。

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class RuntimeControllerSample : MonoBehaviour

{

public AnimationClip clip;

public RuntimeAnimatorController controller;

public float weight;

PlayableGraph playableGraph;

AnimationMixerPlayable mixerPlayable;

void Start()

{

// 创建该图和混合器，然后将它们绑定到 Animator。

playableGraph = PlayableGraph.Create();

var playableOutput = AnimationPlayableOutput.Create(playableGraph,"Animation", GetComponent<Animator>());

mixerPlayable = AnimationMixerPlayable.Create(playableGraph, 2);

playableOutput.SetSourcePlayable(mixerPlayable);

// 创建 AnimationClipPlayable 并将它们连接到混合器。

var clipPlayable = AnimationClipPlayable.Create(playableGraph, clip);

var ctrlPlayable = AnimatorControllerPlayable.Create(playableGraph, controller);

playableGraph.Connect(clipPlayable, 0, mixerPlayable, 0);

playableGraph.Connect(ctrlPlayable, 0, mixerPlayable, 1);

//播放该图。

playableGraph.Play();

}

void Update()

{

weight = Mathf.Clamp01(weight);

mixerPlayable.SetInputWeight(0, 1.0f-weight);

mixerPlayable.SetInputWeight(1, weight);

}

void OnDisable()

{

//销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

## 创建具有若干输出的 PlayableGraph

以下示例演示如何使用以下两个不同的可播放项输出类型来创建 PlayableGraph：AudioPlayableOutput 和 AnimationPlayableOutput。一个 PlayableGraph 可以有许多不同类型的可播放项输出。

此示例还演示如何通过连接到 AudioPlayableOutput 的 AudioClipPlayable 来播放 AudioClip。

using UnityEngine;

using UnityEngine.Animations;

using UnityEngine.Audio;

using UnityEngine.Playables;

[RequireComponent(typeof(Animator))]

[RequireComponent(typeof(AudioSource))]

public class MultiOutputSample : MonoBehaviour

{

public AnimationClip animationClip;

public AudioClip audioClip;

PlayableGraph playableGraph;

void Start()

{

playableGraph = PlayableGraph.Create();

// 创建输出。

var animationOutput = AnimationPlayableOutput.Create(playableGraph,"Animation", GetComponent<Animator>());

var audioOutput = AudioPlayableOutput.Create(playableGraph, "Audio", GetComponent<AudioSource>());

// 创建可播放项。

var animationClipPlayable = AnimationClipPlayable.Create(playableGraph, animationClip);

var audioClipPlayable = AudioClipPlayable.Create(playableGraph, audioClip, true);

//将可播放项连接到输出

animationOutput.SetSourcePlayable(animationClipPlayable);

audioOutput.SetSourcePlayable(audioClipPlayable);

// 播放该图。

playableGraph.Play();

}

void OnDisable()

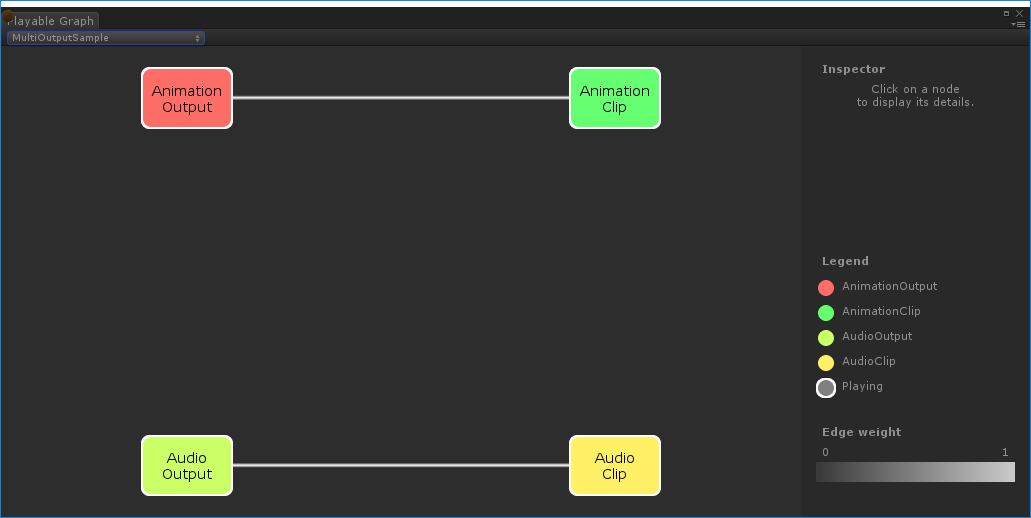
{

//销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

MultiOutputSample 生成的 PlayableGraph

## 控制树的播放状态

以下示例演示如何使用 Playable.SetPlayState() 方法来控制 PlayableGraph 树上节点的播放状态。SetPlayState 方法控制整个树、其分支之一或单个节点的播放状态。

设置节点的播放状态时，状态会传播到所有子节点（无论其播放状态如何）。例如，如果显式暂停了子节点，则将父节点设置为“播放”也会将其所有子节点设置为“播放”。

在此示例中，PlayableGraph 包含的一个混合器将混合两个动画剪辑。AnimationClipPlayable 包裹每个动画剪辑，而 SetPlayState() 方法显式暂停第二个可播放项。第二个 AnimationClipPlayable 被显式暂停，因此其内部时间不会推进而输出相同值。确切值取决于 AnimationClipPlayable 暂停时的具体时间。

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class PauseSubGraphAnimationSample : MonoBehaviour

{

public AnimationClip clip0;

public AnimationClip clip1;

PlayableGraph playableGraph;

AnimationMixerPlayable mixerPlayable;

void Start()

{

// 创建该图和混合器，然后将它们绑定到 Animator。

playableGraph = PlayableGraph.Create();

var playableOutput = AnimationPlayableOutput.Create(playableGraph,"Animation", GetComponent<Animator>());

mixerPlayable = AnimationMixerPlayable.Create(playableGraph, 2);

playableOutput.SetSourcePlayable(mixerPlayable);

// 创建 AnimationClipPlayable 并将它们连接到混合器。

var clipPlayable0 = AnimationClipPlayable.Create(playableGraph, clip0);

var clipPlayable1 = AnimationClipPlayable.Create(playableGraph, clip1);

playableGraph.Connect(clipPlayable0, 0, mixerPlayable, 0);

playableGraph.Connect(clipPlayable1, 0, mixerPlayable, 1);

mixerPlayable.SetInputWeight(0, 1.0f);

mixerPlayable.SetInputWeight(1, 1.0f);

clipPlayable1.SetPlayState(PlayState.Paused);

//播放该图。

playableGraph.Play();

}

void OnDisable()

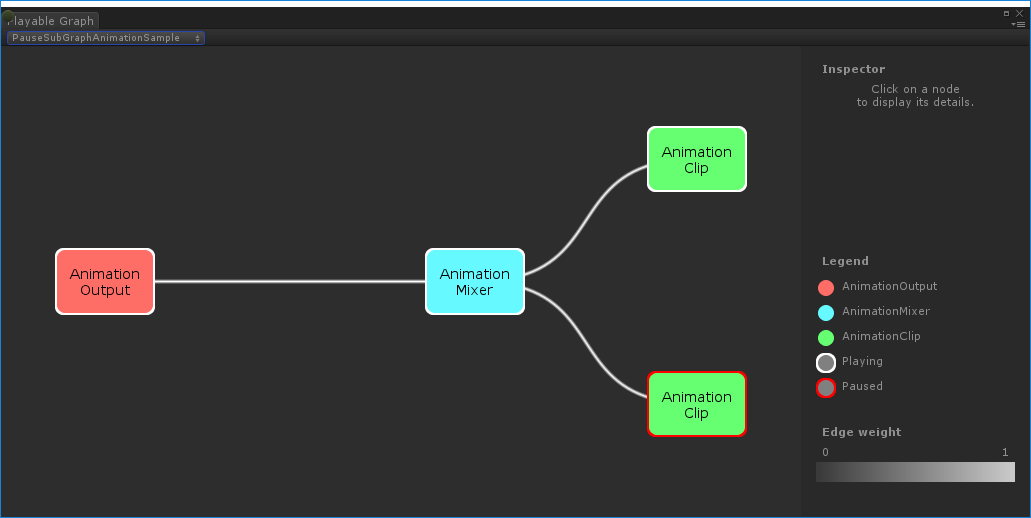
{

//销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

PauseSubGraphAnimationSample 生成的 PlayableGraph。请注意第二个剪辑已暂停（红边）。

## 控制树的时序

以下示例演示如何使用 Play() 方法来播放 PlayableGraph、如何使用 SetPlayState() 方法来暂停可播放项以及如何使用 SetTime() 方法来通过变量手动设置可播放项的本地时间。

using UnityEngine;

using UnityEngine.Playables;

using UnityEngine.Animations;

[RequireComponent(typeof(Animator))]

public class PlayWithTimeControlSample : MonoBehaviour

{

public AnimationClip clip;

public float time;

PlayableGraph playableGraph;

AnimationClipPlayable playableClip;

void Start()

{

playableGraph = PlayableGraph.Create();

var playableOutput = AnimationPlayableOutput.Create(playableGraph, "Animation", GetComponent<Animator>());

// 将剪辑包裹在可播放项中

playableClip = AnimationClipPlayable.Create(playableGraph, clip);

// 将可播放项连接到输出

playableOutput.SetSourcePlayable(playableClip);

// 播放该图。

playableGraph.Play();

//使时间停止自动前进。

playableClip.SetPlayState(PlayState.Paused);

}

void Update ()

{

//手动控制时间

playableClip.SetTime(time);

}

void OnDisable()

{

// 销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

## 创建 PlayableBehaviour

以下示例演示如何使用 PlayableBehaviour 公有类来创建自定义的可播放项。此示例还演示如何重载 PrepareFrame() 虚拟方法以控制 PlayableGraph 上的节点。自定义的可播放项可重载 PlayableBehaviour 类的任何其他虚拟方法。

在此示例中，受控节点是一系列动画剪辑 (clipsToPlay)。SetInputMethod() 将修改每个动画剪辑的混合权重，确保一次只播放一个剪辑，而 SetTime() 方法将调整本地时间，以便在激活动画剪辑时开始播放。

using UnityEngine;

using UnityEngine.Animations;

using UnityEngine.Playables;

public class PlayQueuePlayable : PlayableBehaviour

{

private int m\_CurrentClipIndex = -1;

private float m\_TimeToNextClip;

private Playable mixer;

public void Initialize(AnimationClip[] clipsToPlay, Playable owner, PlayableGraph graph)

{

owner.SetInputCount(1);

mixer = AnimationMixerPlayable.Create(graph, clipsToPlay.Length);

graph.Connect(mixer, 0, owner, 0);

owner.SetInputWeight(0, 1);

for (int clipIndex = 0 ; clipIndex < mixer.GetInputCount() ; ++clipIndex)

{

graph.Connect(AnimationClipPlayable.Create(graph, clipsToPlay[clipIndex]), 0, mixer, clipIndex);

mixer.SetInputWeight(clipIndex, 1.0f);

}

}

override public void PrepareFrame(Playable owner, FrameData info)

{

if (mixer.GetInputCount() == 0)

return;

// 必要时，前进到下一剪辑

m\_TimeToNextClip -= (float)info.deltaTime;

if (m\_TimeToNextClip <= 0.0f)

{

m\_CurrentClipIndex++;

if (m\_CurrentClipIndex >= mixer.GetInputCount())

m\_CurrentClipIndex = 0;

var currentClip = (AnimationClipPlayable)mixer.GetInput(m\_CurrentClipIndex);

// 重置时间，以便下一个剪辑从正确位置开始

currentClip.SetTime(0);

m\_TimeToNextClip = currentClip.GetAnimationClip().length;

}

// 调整输入权重

for (int clipIndex = 0 ; clipIndex < mixer.GetInputCount(); ++clipIndex)

{

if (clipIndex == m\_CurrentClipIndex)

mixer.SetInputWeight(clipIndex, 1.0f);

else

mixer.SetInputWeight(clipIndex, 0.0f);

}

}

}

[RequireComponent(typeof (Animator))]

public class PlayQueueSample : MonoBehaviour

{

public AnimationClip[] clipsToPlay;

PlayableGraph playableGraph;

void Start()

{

playableGraph = PlayableGraph.Create();

var playQueuePlayable = ScriptPlayable<PlayQueuePlayable>.Create(playableGraph);

var playQueue = playQueuePlayable.GetBehaviour();

playQueue.Initialize(clipsToPlay, playQueuePlayable, playableGraph);

var playableOutput = AnimationPlayableOutput.Create(playableGraph, "Animation", GetComponent<Animator>());

playableOutput.SetSourcePlayable(playQueuePlayable);

playableOutput.SetSourceInputPort(0);

playableGraph.Play();

}

void OnDisable()

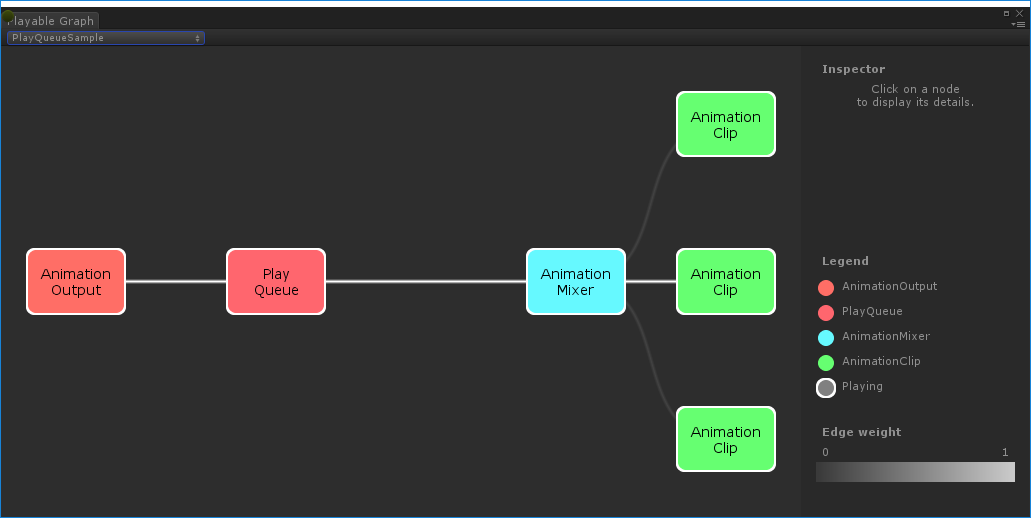
{

// 销毁该图创建的所有可播放项和输出。

playableGraph.Destroy();

}

}

PlayQueueSample 生成的 PlayableGraph