[TOC]

简体中文

Unity音频池系统

概述

本系统实现了Unity中非绑定物体音频播放的池化，给音频设计师提供了：

1. 可选择的音频实例回收功能
2. 可自定义的平台特化音频实例数规则
3. 统一且解耦的AudioManager单例类，用于提供音频播放入口
4. AudioEvent类，提供可扩展的音频单例基类

使用说明

AudioConfig

`AudioConfig` 定义了音频相关的设置，设计师可以通过`Asset (folder) → Scriptable Object → AudioConfig` 创建一个Scriptable Object。

字段

该可编程物体包含了2个***公共字段***:

1. `AudioRecycleStrategy recycleStrategy` 用于定义音频实例的回收策略。本系统提供了3种回收策略，`ByTime`, `ByVolume`, 和`ByDistance`。
   `ByTime` 允许系统按音频播放时间顺序回收实例（按最早开始播放 → 最晚开始播放的顺序） `ByVolume` 允许系统按音频音量排序回收实例（按最小音量 → 最大音量），如果音量相同，随机按默认顺序回收。
   `ByDistance` 允许系统按音频实例距离玩家位置远近回收（按最远 → 最近），如果距离相同，随机按默认顺序回收。

2. `List<PlatformObjectNumber> objectPerPlatform`
   该列表允许设计师在Unity检视器内针对各平台设置最大音频实例数。 `struct PlayformObjectNumber` 标识了特定平台可以拥有的最大音频实例数。设计师可以在此处选择多种平台以匹配同一个实例数。

方法

该可编程物体还包含了1个***公共方法***:

```
public int GetMaxNumber();
```

该方法会自动返回已设置的当前平台可持有的最大实例数。

```
示例

设计师设置:
    Android & iPhone => 10
    Windows & Linux => 30
    PS4 & PS5 => 25

当前运行平台:
    Windows

GetMaxNumber():
    返回 30
```

AudioManager

`AudioManager` 是一个单例类。设计师可以通过提供配置好的`AudioConfig`使其按预定方案处理音频相关的事件。

字段

`AudioManager` 持有`2`个***公共字段***:

1. `AudioConfig config` 用于配置当前场景内的音频相关设定。设计师可以直接在Unity检视器内将设置好的`AudioConfig`拖入该字段。

2. `AudioEvent audioEventPrefab` 用于生成音频播放实例。设计师可以将任意挂载了`AudioEvent`组件的预制体拖入该字段。

方法

`AudioManager` 持有`4`个***公共方法***:

1. **播放**

```
public AudioEvent Play(
        AudioClip clip,
        Vector3 position,
        bool loop = false,
        float volume = 1f,
        float pitch = 1f,
        float spatialBlend = 1f,
        AudioMixerGroup mixerGroup = null);
```

该方法提供了使用AudioManager播放音频的入口。设计师在调用该方法时，需要传入一个AudioClip和Vector3。其他参数已设置默认值，设计师可按需求填入。

该方法返回了一个AudioEvent实例，以供处理音频控制。

> 在使用该方法时，推荐使用一个AudioEvent变量保存持有返回的实例。

2. **停止播放**

```
public void Stop(AudioEvent audioEvent);
```

该方法提供了使用AudioManager停止正在播放音频的入口。设计师在调用该方法时，需要传入一个AudioEvent。

对于非循环音频，系统会自动管理，并在音频播放结束时回收。尽管如此，设计师也可以通过调用该方法提前结束并回收非循环音频。

3. **手动回收音频实例**

```
public void ApplyRecycle();
```

该方法提供了使用AudioManager手动回收当前音频实例的入口。

该方法将按照前文描述的，已设置好的AudioRecycleStrategy策略回收1个AudioEvent实例。

> 该公共方法被提供给设计师以提供最大限度的自由。尽管如此，并不建议设计师在脚本中单独调用该方法。事实上，`Play()` 方法的实现中已自动管理了这些实例。

4. **手动设置AudioRecycleStrategy**

```
// 设置到指定策略
public void ResetStrategy(AudioRecycleStrategy strategy);
```

```
// 重置为Audio Config的策略
public void ResetStrategy();
```

该方法提供了使用AudioManager手动设置音频回收策略的入口。使用该入口，设计师可以在运行时更改AudioManager的音频回收策略。

该方法有2个重载。设计师可以通过传入AudioRecycleStrategy来设置到指定策略，或调用无参数重载以重置到AudioConfig中指定的策略。

使用示例

```
public AudioClip clip; // 需要播放的音频
public Vector3 position; // 音频播放位置
public bool loop = true; // 循环音频

private AudioEvent _e; // 私有AudioEvent字段用于存储实例引用

public void ExampleCallToPlay() // 播放音频
{
    _e = AudioManager.Instance.Play(
        clip,
        position,
        loop);
}

public void ExampleCallToStop() // 停止该音频的播放
{
    AudioManager.Instance.Stop(_e);
}
```

AudioEvent

AudioEvent 类负责音频的具体播放行为。当AudioManager.Play(...) 被调用时，该类的实例被创建。

AudioEvent 会强制绑定一个AudioSource 到挂载的游戏物体上。该AudioSource 组件负责播放音频。

该类实现了IPoolable 接口，代表它是可对象池化的类。

字段

`AudioEvent` 持有2个***公共字段***:

1. `OnActive` 该事件在`AudioEvent` 实例激活时被广播。其他脚本可以订阅该事件以在**音频播放**时获得通知。
2. `OnDeactive` 该事件在`AudioEvent` 实例禁用时被广播。其他脚本可以订阅该事件以在**音频停止**时获得通知。

方法

`AudioEvent` 持有8个***公共方法***:

1. **被激活时**

```
public void OnSpawn();
```

`IPoolable` 接口的实现。该方法在该实例被从对象池中激活时被调用。

该方法目前为空。

2. **被禁用时**

```
public void OnDespawn();
```

`IPoolable` 接口的实现。该方法在该实例被禁用并放置回对象池时被调用。

该方法目前重置所有`AudioSource` 参数。

3. **播放**

```
public void Play(AudioClip clip,
        Vector3 pos,
        bool loop = false,
        float volume = 1f,
        float pitch = 1f,
        float spatialBlend = 1f,
        AudioMixerGroup mixerGroup = null);
```

该方法将播放传入的音频，并按照传入的参数配置`AudioSource`。

> 尽管提供了该方法，设计师应使用`AudioManager.Play(...)`，而非`AudioEvent.Play(...)`。

4. 停止音频

```
public void Stop();
```

该方法将停止该实例正在播放的音频。

> 尽管提供了该方法，设计师应使用`AudioManager.Stop(AudioEvent)`，而非`AudioEvent.Stop()`。

5. 获取当前音频

```
public AudioClip GetAudioClip();
```

该方法返回当前正在播放的音频引用。

6. 获取当前位置

```
public Vector3 GetPosition();
```

该方法返回当前实例的位置。

6. 获取当前音量

```
public float GetVolume()
```

该方法返回当前实例的播放音量。

6. 获取当前播放状态

```
public bool IsPlaying()
```

该方法返回一个布尔值，`true` 当音频正在播放， `false` 当音频已停止。

总结

该系统提供了一个完整的音频对象池系统的实现。重点构建了一个`AudioManager`单例类用于提供统一的音频管理接口。 音频设计师应当使用`AudioManager`提供的公共方法，以确保游戏架构**解耦**，**可扩展**，**易维护**。

ENGLISH

Unity Audio Pooling System

Overview

This system implements pooling for non-attached audio playback in Unity, providing audio designers with:

1. Multiple audio instance recycling functionalities to choose from
2. Customizable platform-specific audio instance count rules
3. A unified and decoupled AudioManager singleton class for providing audio playback entry points
4. An AudioEvent class that serves as an extensible audio instance base class

Usage Instructions

AudioConfig

`AudioConfig` defines audio-related settings. Designers can create a Scriptable Object by navigating to `Asset (folder) → Scriptable Object → AudioConfig`.

Fields

This programmable object contains `2` ***public fields***:

1. `AudioRecycleStrategy recycleStrategy` is used to define the audio instance recycling strategy. This system provides `3` recycling strategies: `ByTime`, `ByVolume`, and `ByDistance`. `ByTime` allows the system to recycle instances in the order of **audio playback time** (from earliest start to latest start). `ByVolume` allows the system to recycle instances based on **audio volume** (from lowest to highest volume). If volumes are the same, recycling follows a random default order. `ByDistance` allows the system to recycle instances based on their **distance from the player** (from

farthest to nearest). If distances are the same, recycling follows a random default order.

2. `List<PlatformObjectNumber> objectPerPlatform`
   This list allows designers to set the **maximum number of audio instances per platform** in the Unity inspector. The `struct PlatformObjectNumber` identifies the maximum number of audio instances allowed for specific platforms. Designers can select multiple platforms to match the same instance count here.

Methods

This programmable object also contains `1` ***public method***:

```
public int GetMaxNumber();
```

This method automatically returns the maximum number of instances allowed for the `current platform`.

```
Example

Designer set:
    Android & iPhone => 10
    Windows & Linux => 30
    PS4 & PS5 => 25

Current platform:
    Windows

GetMaxNumber():
    Returns 30
```

AudioManager

`AudioManager` is a singleton class. Designers can provide a configured `AudioConfig` to have it handle audio-related events according to the predetermined scheme.

Fields

`AudioManager` holds `2` ***public fields***:

1. `AudioConfig config` is used to configure audio-related settings in the current scene. Designers can directly drag a configured `AudioConfig` into this field in the Unity inspector.

2. `AudioEvent audioEventPrefab` is used to generate audio playback instances. Designers can drag any prefab with an `AudioEvent` component attached into this field.

Methods

`AudioManager` holds `4` ***public methods***:

1. **Play**

```
public AudioEvent Play(
        AudioClip clip,
        Vector3 position,
        bool loop = false,
        float volume = 1f,
        float pitch = 1f,
        float spatialBlend = 1f,
        AudioMixerGroup mixerGroup = null);
```

This method provides an entry point for using `AudioManager` to play audio. When calling this method, designers need to pass in an `AudioClip` and a `Vector3`. Other parameters have default values, and designers can fill them in as needed.

This method returns an `AudioEvent` instance for handling audio control.

> When using this method, it is recommended to use an `AudioEvent` variable to hold the returned instance.

1. **Stop**

```
public void Stop(AudioEvent audioEvent);
```

This method provides an entry point for using `AudioManager` to stop currently playing audio. When calling this method, designers need to pass in an `AudioEvent`.

For non-looping audio, the system automatically manages and recycles the instance when the audio finishes playing. However, designers can also call this method to stop and recycle non-looping audio early.

3. **Manually Recycle Audio Instances**

```
public void ApplyRecycle();
```

This method provides an entry point for using `AudioManager` to manually recycle current audio instances.

This method will recycle `1` `AudioEvent` instance according to the previously described and set `AudioRecycleStrategy`.

> This public method is provided to designers to offer maximum freedom. However, it is ***NOT*** recommended for designers to call this method individually in scripts. In fact, the `Play(...)` method implementation already automatically manages these instances.

4. **Manually Set `AudioRecycleStrategy`**

```
// Set to a specified strategy
public void ResetStrategy(AudioRecycleStrategy strategy);

// Reset to the strategy specified in Audio Config
public void ResetStrategy();
```

This method provides an entry point for using `AudioManager` to manually set the audio recycle strategy. Using this entry point, designers can change the audio recycle strategy of `AudioManager` at runtime.

This method has `2` overloads. Designers can set to a specified strategy by passing in an `AudioRecycleStrategy`, or call the parameterless overload to reset to the strategy specified in `AudioConfig`.

Usage Example

```
public AudioClip clip; // Audio to be played
public Vector3 position; // Audio playback position
public bool loop = true; // Looping audio

private AudioEvent _e; // Private AudioEvent field to store instance
reference

public void ExampleCallToPlay() // Play audio
{
    _e = AudioManager.Instance.Play(
        clip,
        position,
        loop);
}

public void ExampleCallToStop() // Stop audio playback
{
    AudioManager.Instance.Stop(_e);
}
```

AudioEvent

`AudioEvent` class is responsible for the specific playback behavior of audio. When `AudioManager.Play(...)` is called, an instance of this class is created.

`AudioEvent` will enforce the binding of an `AudioSource` to the game object it is attached to. This `AudioSource` component is responsible for playing the audio.

This class implements the `IPoolable` interface, indicating that it is a poolable object.

Fields

`AudioEvent` holds `2` ***public fields***:

1. `OnActive` This event is broadcast when the `AudioEvent` instance is activated. Other scripts can subscribe to this event to be notified when **audio playback** occurs.
2. `OnDeactive` This event is broadcast when the `AudioEvent` instance is deactivated. Other scripts can subscribe to this event to be notified when **audio stops**.

Methods

`AudioEvent` holds `8` ***public methods***:

1. **On Spawn**

```
public void OnSpawn();
```

`IPoolable` interface implementation. This method is called when the instance is activated from the object pool.

This method is currently empty.

2. **On Despawn**

```
public void OnDespawn();
```

`IPoolable` interface implementation. This method is called when the instance is deactivated and returned to the object pool.

This method currently resets all `AudioSource` parameters.

3. **Play**

```
public void Play(AudioClip clip,
        Vector3 pos,
        bool loop = false,
        float volume = 1f,
        float pitch = 1f,
        float spatialBlend = 1f,
        AudioMixerGroup mixerGroup = null);
```

This method will play the passed-in audio and configure the `AudioSource` according to the passed-in parameters.

> Although this method is provided, designers should use `AudioManager.Play(...)` instead of `AudioEvent.Play(...)`.

4. **Stop**

```
public void Stop();
```

This method will stop the audio currently playing on this instance.

> Although this method is provided, designers should use `AudioManager.Stop(AudioEvent)` instead of `AudioEvent.Stop()`.

5. **Get Audio Clip**

```
public AudioClip GetAudioClip();
```

This method returns the reference to the audio currently playing.

6. **Get Position**

```
public Vector3 GetPosition();
```

This method returns the position of the current instance.

6. **Get Volume**

```
public float GetVolume()
```

This method returns the playback volume of the current instance.

6. **Get Playing Status**

```
public bool IsPlaying()
```

This method returns a boolean value, `true` when the audio is playing, and `false` when the audio has stopped.

Summary

This system provides a complete implementation of an audio object pool system. It focuses on constructing a singleton class `AudioManager` to provide a unified audio management interface. Audio designers should use the public methods provided by `AudioManager` to ensure the game architecture is **decoupled**, **extensible**, and **maintainable**.

由李江浩创作 Created by Jianghao Li