GameFeedback框架的核心部分，主要由两个部分组成，核心库和通用工具库。

## 核心库

核心库包含框架最重要的三个类：

- [GameFeedback](#GameFeedback)

- [GameFeedbackEffect](#GameFeedbackEffect)

- [GameFeedbackPlayer](#GameFeedbackPlayer)

## GameFeedback

```cpp

UCLASS(BlueprintType, Blueprintable, EditInlineNew)

class GAMEFEEDBACKCORE\_API UGameFeedback : public UObject

{

GENERATED\_BODY()

///...

}

```

GameFeedback类主要承担效果装配的作用，每个GameFeedback对象都维护着一个GameFeedbackEffect列表，GameFeedback本身没有附带任何实际效果，所有的效果都由GameFeedbackEffect对象实现，GameFeedback只是一个状态机，负责调度、更新其维护的GameFeedbackEffect对象。

##### 生命周期

GameFeedback对象的状态总共有四个，分别为：

- NotInitialized,

- Idle

- Running

- Paused

其中，状态之间的转换关系如下图：

```mermaid

stateDiagram-v2

[\*] --> NotInitialized

NotInitialized --> Idle : InitFeedback()

Idle --> Running : PlayFeedback()

Running --> Idle : StopFeedback()

Running --> Paused : PauseFeedback()

Paused --> Running : ResumeFeedback()

Paused --> Idle : StopFeedback()

```

##### 事件

GameFeedback对象在状态变化时会触发相应的事件，这些事件可以被绑定的GameFeedbackPlayer监听和处理。

一般情况下，这些事件为插件内部使用，用户无需关注，但若有一些特殊的需求，也可自行监听和处理。

###### OnGameFeedbackStateChanged

```cpp

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FOnGameFeedbackStateChanged, EGameFeedbackState, NewState);

UPROPERTY(BlueprintAssignable, Category = "GameFeedback")

FOnGameFeedbackStateChanged OnGameFeedbackStateChanged;

```

该事件在GameFeedback自身状态发生改变时广播，事件将广播GameFeedback当前的状态。

###### OnGameFeedbackStopped

```cpp

DECLARE\_DYNAMIC\_MULTICAST\_DELEGATE\_OneParam(FOnGameFeedbackStopped, bool, bInterrupted);

UPROPERTY(BlueprintAssignable, Category = "GameFeedback")

FOnGameFeedbackStopped OnGameFeedbackStopped;

```

该事件在GameFeedback停止时广播，值得注意的是，它将先于OnGameFeedbackStateChanged事件广播，事件将广播当前GameFeedback停止时是否属于被打断的情况。

##### 主要函数

```cpp

void InitFeedback();

void PlayFeedback();

void PauseFeedback();

void ResumeFeedback();

void StopFeedback();

void TickFeedback(float DeltaTime);

```

###### InitFeedback()

用于初始化GameFeedback，除了设置自身状态之外，函数将遍历自身维护的GameFeedbackEffect，并调用GameFeedbackEffect的初始化函数。

###### PlayFeedback()

用于播放GameFeedback，调用后，Feedback将标记自身为Running状态，但其自身并没有Tick的能力，Feedback的Tick需要由其绑定的GameFeedbackPlayer来进行。

GameFeedback实现了事件驱动，仅在状态发生变化的时候通知其绑定的GameFeedbackPlayer。设置状态的函数如下：

```cpp

void UGameFeedback::SetState(const EGameFeedbackState NewState)

{

if (State != NewState)

{

State = NewState;

OnGameFeedbackStateChanged.Broadcast(State);

}

}

```

###### PauseFeedback()

暂停播放中的GameFeedback，该方法同样仅标记自身状态为暂停状态，并遍历自身维护的GameFeedbackEffect，调用其暂停方法，自身并没有暂停Tick的能力。包括后面的ResumeFeedback()方法，都仅改变自身状态，并广播状态变更事件，具体的与Tick相关的事务将由其绑定的GameFeedbackPlayer来维护。

###### ResumeFeedback()

继续暂停中的GameFeedback，具体原理同上。

###### StopFeedback()

停止GameFeedback，该函数存在两种调用情况，一种由GameFeedback维护，当所有维护的GameFeedbackEffect都播放完毕之后，自动调用，此时GameFeedback属于正常执行完成的停止，而非被打断。

而当GameFeedback仍处于Running或Paused状态，用户自行调用了StopFeedback()方法时，当前Feedback将被强行终止，并且将被标识为被打断的停止操作。GameFeedbackEffect可以获取到停止操作是否为被打断的状态，并作出不同的处理。利用这点我们可以确保效果的打断安全、准确。

###### TickFeedback(float DeltaTime)

GameFeedback的Tick函数，由GameFeedbackPlayer调用和维护，用户不应该调用该函数。

该Tick函数负责更新GameFeedbackEffect和维护GameFeedback自身状态，在每一次Tick，GameFeedback都将按照GameFeedbackEffect列表的顺序，逐个调用GameFeedbackEffect的Tick函数。GameFeedbackEffect也有自身的状态，当所有GameFeedbackEffect都播放结束之后，GameFeedback才会自动结束。

GameFeedbackEffect的Tick顺序没有优先级，这样设计是为了使效果的设计逻辑更加直观统一，效果的执行始终保持从上到下依次执行，而不存在跳跃式的执行顺序。

## GameFeedbackEffect

```cpp

UCLASS(Abstract, EditInlineNew, AutoExpandCategories=(Basic))

class GAMEFEEDBACKCORE\_API UGameFeedbackEffectBase : public UObject

{

GENERATED\_BODY()

///...

}

```

GameFeedbackEffect类处理实际的效果逻辑，这些由子类实现，子类定义了具体的行为。对于一个继承了GameFeedbackEffect类的子类，需要关注的函数如下：

```cpp

// Life cycle

virtual void OnInit() {}

virtual void OnPlay() {}

virtual void OnPause() {}

virtual void OnResume() {}

virtual void OnStop(bool bInterrupted) {}

virtual void OnTick(float DeltaTime) {}

// Editor

virtual EGameFeedbackEffectType GetEffectType() const

{

return EGameFeedbackEffectType::Custom;

}

```

以上生命周期回调名称基本上已经说明清楚了它们各自的功能，子类可以重写这些虚函数并插入自己效果的逻辑。GameFeedbackEffect的状态基本上跟GameFeedback是一样的，因为GameFeedback其实也就是做了一下转发而已，下图是GameFeedbackEffect的状态图：

```mermaid

stateDiagram-v2

[\*] --> NotInitialized

NotInitialized --> Idle : Init()

Idle --> Running : Play()

Running --> Idle : Stop()

Running --> Paused : Pause()

Paused --> Running : Resume()

Paused --> Idle : Stop()

```

GameFeedbackEffect类声明了一个`EGameFeedbackEffectType GetEffectType()`函数，里面返回了一个`EGameFeedbackEffectType`枚举，该枚举变量定义了GameFeedbackEffect的类型，该枚举仅用于编辑器，主要作为标记供自定义属性编辑器（尚未实现）使用。

## GameFeedbackPlayer

```cpp

UCLASS()

class GAMEFEEDBACKCORE\_API UGameFeedbackPlayer : public UObject, public FTickableGameObject

{

GENERATED\_BODY()

}

```

GameFeedbackPlayer类负责处理GameFeedback，它实现了FTickableGameObject，具有Tick的功能。

要播放一个GameFeedback，首先需要创建一个GameFeedbackPlayer对象，并调用Player对象的LoadFeedback方法，Load完成后再调用Player的PlayerFeedback函数。相关函数如下：

```cpp

public:

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void LoadFeedback(UGameFeedback\* Feedback);

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void PlayFeedback(bool bUseAutoUnload = false);

```

另外，GameFeedbackPlayer还提供了用于控制GameFeedback的方法，用户不应该直接调用GameFeedback中的方法，而应一律使用GameFeedbackPlayer来操作GameFeedback。

```cpp

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void PauseFeedback() const;

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void ResumeFeedback() const;

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void StopFeedback() const;

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void ReplayFeedback() const;

UFUNCTION(BlueprintCallable, Category = "GameFeedbackPlayer")

void UnloadFeedback();

```