cokatielシステム

SmartAudio16

ver. 1.4.0

プログラマーズガイド

2014.02.26

株式会社ピュアサウンド

目次

1.はじめに

この文書では、『cockatielシステム SmartAudio16(以下SmartAudio16)』の典型的な使用方法を記載します。SmartAudio16の開始から演奏、終了までの各プロセスを、具体的な操作手順を説明しながら紹介していきます。

SmartAudio16には、iOS上ではobjective-C用に、AndroidOS上ではJava用に用意されたクラス『CktlDriver』をインターフェースとしてアクセスします。
このプログラマーズガイド内に記載されているメソッドは、特別に説明がない限りすべて『CktlDriver』クラスのメソッドです。
なお、AndroidOSでNativeActivityを使用する場合はJavaのクラスを使用せず、NativeActivityのために用意されたC言語の関数をインターフェースとしてSmartAudio16にアクセスします。

NativeActivityでのアプリケーション開発を行っている場合には、別紙『NativeActivityガイド』を参照してください。

2.準備

SmartAudio16をプロジェクトで使用するためには、ライブラリ本体、ライブラリヘッダ(Javaの場合はクラスライブラリ)、リソースファイルの三つのファイルが必要です。
ライブラリ本体とライブラリヘッダ(Javaの場合はクラスライブラリ)はiOSとAndroidOSでそれぞれ独自のものを使用します。
リソースファイルはiOSとAndroidOSで共通の『spkgサウンドパッケージファイル(以下サウンドパッケージファイル)』を使用します。

2.1.iOSでの準備

iOSでは、次の三つのファイルを使用します。

ライブラリ本体： libSmartAudio16.a(静的ライブラリファイル)(注)
ライブラリヘッダ： SmartAudio16_iOS.h
リソースファイル： spkgサウンドパッケージファイル

以下に、Xcodeでプログラミングをする場合の手順を示します。

1) ライブラリ本体、ライブラリヘッダ、リソースファイルをプロジェクトに登録します。

2) 次にプロジェクト設定画面から『TARGETS』を選択し、さらに『Build Phase』を選択します。
『Link Binary With Libraries』を開いて内容を確認し、『libSmartAudio16.a』が表示されていれば問題ありません。
登録されていない場合には、プロジェクトのファイル一覧から『libSmartAudio16.a』をドラッグ&ドロップして登録します。

3) SmartAudio16は、オーディオの処理に『AudioToolbox』フレームワークを使用しています。
そのため、『Link Binary With Libraries』に『AudioToolbox.framework』を追加登録します。

ここまでで、プロジェクト上の設定は完了です。

4) ここからは、ソースコードの記述を行ないます。
『ViewController.h』など、プロジェクトのメインとして使用するクラスが定義されているヘッダファイルに『SmartAudio16_iOS.h』をインクルードします。

```
#include "SmartAudio16_iOS.h"
```

5) 最後に、メインとなるクラスの宣言部にCktlDriver型のポインタを宣言します。

```
@interface ViewController : UIViewController {
    CktlDriver        *cktlDriver;
}
```

これで、SmartAudio16をプロジェクト内で使用する準備が整いました。

(注)iOS版では、実機用のライブラリファイルとPCシミュレータ用のライブラリファイルは異なりますので、注意してください。
このプログラマーズガイドでは、実機用のライブラリファイルを使用するものとして説明を行います。

また、本ライブラリはarmv7アーキテクチャのみをターゲットとしています。armv6アーキテクチャ用のライブラリファイルが必要な場合は、別途ご相談ください。

2.2.AndroidOS(Java)での準備

AndoroidOS(Java)では、次の三つのファイルを使用します。

ライブラリ本体: libSmartAudio16.so(動的ライブラリファイル)
クラスライブラリ: SmartAudio16.jar
リソースファイル: spkgサウンドパッケージファイル

1) ライブラリ本体、クラスライブラリ、リソースファイルをプロジェクトに登録します。

[フォルダ配置例]

```
PSSsample/
        +--- assets ……ここにサウンドパッケージファイルを登録します。
        +--- libs/
                +--- SmartAudio16.jar
                +--- armeabi/
                        +--- libSmartAudio16Library.so
```

2) ここからは、ソースコードの記述を行ないます。
メインとなるソースファイル内にクラスライブラリをインポートするための記述をします。

```
import smartaudio16.library.CktlDriver;
```

3) 次に、CktlDriverクラスのインスタンスを作成します。

```
CktlDriver mPSS = new CktlDriver(this);
```

引数にはactivityを継承した親クラスを指定します。通常は、『this』キーワードを指定します。

これで、SmartAudio16をプロジェクト内で使用する準備が整いました。

3.SmartAudio16の初期化

準備が完了したら、実際にSmartAudio16を使用するために初期化を行います。
初期化をするためには、次のメソッドを呼び出します。

iOS

```
-(id)cktlInit:(UInt32) heapSize
  playerCount:(SInt8) player
soundFilePath:(NSString *)filePath
    debugFlag:(BOOL) debugFlag;
```

AndroidOS

```
int cktlInit(int     heapSize,
             byte    player,
             String  filePath,
             boolean debugFlag);
```

第一引数：heapSize

アプリケーションで使用するサウンドヒープのサイズを指定します。
サウンドヒープの詳細は、"SmartAudio16：イントロダクション"ドキュメント内の"7.メモリ管理について"に記載されています。

第二引数：player

使用するシーケンスプレイヤーの最大数を指定します。
1から16までの値が指定可能です。
シーケンスプレイヤーの詳細は、"SmartAudio16：イントロダクション"ドキュメント内の"5.シーケンスプレイヤーについて"に記載されています。

第三引数：filePath

サウンドパッケージファイルの格納場所を、絶対パスで記述します。
AndroidOSの場合はspkgサウンドパッケージファイルが格納されている任意の絶対パスを、iOSの場合はリソースとしてプロジェクトに登録したspkgサウンドパッケージファイルの絶対パスを指定します。

第四引数：debugFlag

デバッグ用ログ出力の有無を指定します。
有効(YES もしくは true)にすると、コンソールにデバッグ情報が出力されるようになります。
ただし実機のみでの実行時には、有効にしていても出力されません。

iOS版では、ここではじめてCktlDriverクラスのインスタンスが作成されることとなります。
AndroidOS版では、すでに『1.準備』の項でCktlDriverクラスのインスタンスを作成しているため、ここで行われるのはSmartAudio16の初期化のみです。

また、この段階で定期的にオーディオ処理を行うためのオーディオコールバック用のスレッドが作成されます。このスレッドはアプリケーションの終了、もしくはCktlDriveクラスのインスタンスを解放してSmartAudio16を終了した時点で破棄されます。

なお、この初期化用メソッドを呼び出す場所は任意ですが、次項以降で説明するCktlDriverクラスの他のメソッドより先に呼び出しておく必要があります。

3.1.初期化時メモリサイズの目安

初期化メソッドcktlInitを呼び出すと、SmartAudio16が常に必要とする情報を、サウンドパッケージファイルから読み取り、サウンドヒープ内の先頭に保存します。
この情報を、『レジデントインフォメーション』と呼びます。
レジデントインフォメーションには、オーディオデータ、インストゥルメンツ、シーケンス、グループ、コールラベルそれぞれの定義情報全てと、サウンドパッケージファイルの絶対パス情報と、シーケンスプレイヤーのワークエリアが含まれます。

レジデントインフォメーションは、SmartAudio16が終了するまでサウンドヒープ内に保持されます。
したがってサウンドヒープのサイズは最低限、レジデントインフォメーションが格納できるだけのサイズを確保する必要があります。

レジデントインフォメーションの内訳

	必要とするサイズ
spkgファイルパス	不定(文字数分)※
シーケンスプレイヤー	1696byte×シーケンスプレイヤー数
グループ・ラベル名	不定(文字数分)※
オーディオデータ	40byte×オーディオデータ数
インストゥルメンツ	20byte×インストゥルメンツ数
シーケンス	20byte×シーケンス数
グループ	12byte×グループ数
コールラベル	16byte×コールラベル数

※アライメントの関係上、必ず4の倍数になります。

なお、上記の表はレジデントインフォメーションのみのメモリサイズです。
これに加えて、サウンドデータ(オーディオデータおよびシーケンスデータ)の一部もしくは全部を格納できるだけのメモリサイズが必要になります。
サウンドデータのために確保するサイズをどの程度にするかは、アプリケーションの設計方針によります。

4.演奏に必要なデータのロード

SmartAudio16の初期化が完了したら、次にシーケンスデータと、シーケンスデータを演奏するために必要なオーディオデータをサウンドヒープにロードします。
一個もしくは複数個のシーケンスデータおよびオーディオデータの集まりが、ひとつの『グループ』としてサウンドデータ内に定義されています。
サウンドヒープへのデータロードは、『グループ』を単位として行われます。
なお初期化から演奏開始までの間であれば、ロードするタイミングは自由です。

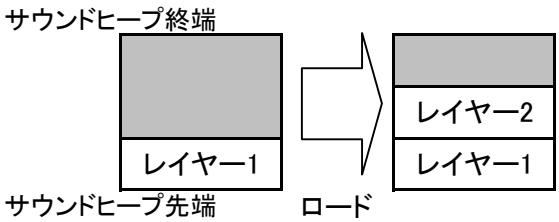
```
-(SInt32)cktlGroupLoad:(UInt16) groupNo;
```
iOS

```
    int cktlGroupLoad(short groupNo);
```
AndroidOS

第一引数：groupNo

ロードするグループの番号を指定します。
成功すると、引数で指定したグループに含まれるコールラベルが割り当てられたシーケンスのデータ、ならびにそのシーケンスデータを演奏するために必要なオーディオデータが、一括でサウンドヒープにロードされます。

グループは、積み重ねてロードすることができ、その積み重ねた位置を『レイヤー』と呼びます。
グループをロードするたび、新たなレイヤーとしてサウンドヒープ内に積み重ねられていきます。
サウンドヒープ内には、最大255個のグループを積み重ねることができます。



新たにグループをロードすると、層のように積み重ねられていきます。

なお、ロード対象のグループに含まれるシーケンスデータやオーディオデータがすでにサウンドヒープ内に存在している場合は、これらのシーケンスデータやオーディオデータが重複してロードされることはありません。
また詳細は後述しますが、グループの削除は最上位のレイヤーに置かれたグループが対象となります。
そのため、グループをロードする順番はアプリケーションの仕様にあわせて適切に管理する必要があります。

5.シーケンスの演奏

グループのロードに成功したら、グループに含まれるシーケンスを演奏をしてみましょう。
演奏を開始するためには、次のメソッドを呼び出します。

iOS

```
-(SInt32)cktlStartSequence:(UInt16)labelNo
                    volume:(SInt8)volume
                       pan:(SInt8)pan;
```

AndroidOS

```
    int cktlStartSequence(short labelNo,
                          byte  volume,
                          byte  pan);
```

第一引数：labelNo

演奏したいシーケンスを割り当てているコールラベル番号を指定します。

第二引数：volume

演奏開始時の音量を0～127の範囲で指定します。

第三引数：pan

演奏開始時の定位を-64～63の範囲で指定します。
-64が最も左寄り、0が中央、63が最も右寄りとなります。

メソッドの呼び出しに成功すると、空いているもしくは最もプライオリティの低いシーケンスプレイヤーを用いて、演奏が開始されます。
戻り値は使用されたシーケンスプレイヤーの番号ですが、これは後述する演奏の停止や、音量や音程の変更などに用います。
なお該当シーケンスデータがサウンドヒープ内に存在していない場合、引数に不正な値を入力した場合は、演奏失敗として処理され、音は鳴りません。

6.シーケンスの停止

『5.シーケンスの演奏』にて演奏を開始したシーケンスは、次のメソッドを呼び出すことで停止することができます。

iOS

```
-(SInt32)cktlStopSequence:(UInt8)playerNo
                fadeOutMsec:(SInt16)fadeOutMsec;
```

AndroidOS

```
    int cktlStopSequence(byte  playerNo
                         short fadeOutMsec);
```

第一引数：playerNo

演奏を停止したいシーケンスプレイヤー番号を指定します。

シーケンスプレイヤー番号は、演奏開始メソッドの戻り値として取得することができます。

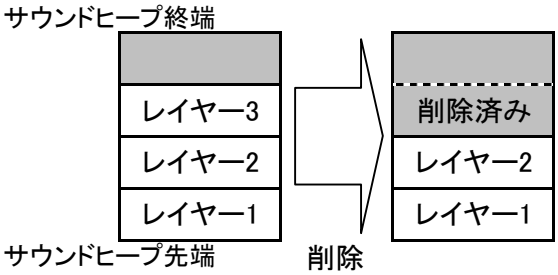
第二引数：fadeOutMsec

このメソッドを呼び出してから、完全に音量が0となるまでのフェードアウト時間を、ミリ秒単位で指定することができます。

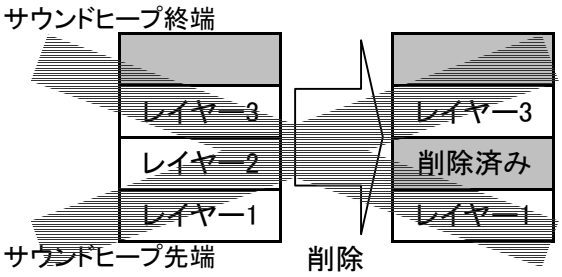
0を指定すると、フェードアウトせず直ちに演奏を停止します。

7.グループの削除

不要になったグループを削除することにより、サウンドヒープを有効活用することができます。
ただし、削除可能なグループは最上位のレイヤーに置かれたグループのみです。



最上位のレイヤーに置かれたグループのみ削除可能です。



最上位のレイヤー以外のグループを直接指定して削除することは出来ません。

次のメソッドを呼び出すことで、最上位のレイヤーに置かれたグループを削除することが出来ます。

```
-(SInt32)cktlFreeTopLayer;
```
iOS

```
    int cktlFreeTopLayer(void);
```
AndroidOS

削除の対象は最上位のレイヤーに置かれたグループ固定のため、引数はありません。
グループの削除に失敗した場合、メソッドは-1を返します。

8.割り込み対応について

SmartAudio16を使用するにあたって、着信などの割り込みに対する適切な処置が必要となります。
ここでは、割り込み発生時の状態の変化や、変化に応じて行なうべき処理を記載します。

8.1.通常時の挙動

- **iOSの場合**

  例えばヘッドフォンを接続した際にスピーカーからの発音を行なうかなど、SmartAudio16での音声出力は、アプリケーションのAudioSession設定に準拠します。
  AudioSessionに関しては、アップル社のマニュアルを参照してください。

- **AndroidOSの場合**

  割り込みが発生していない状態では、特に留意する点はありません。

8.2.割り込み発生時の挙動とその対応

- **iOSの場合**

  HOMEボタンを押してアプリケーションがバックグラウンド状態に移行する場合、シーケンスの演奏はフェードアウトして停止し、音声出力も停止します。
  再びアプリケーションをフォアグラウンド状態にすることで、停止したときの続きから演奏および音声出力が再開します。
  電話やメールなどの着信でダイアログが開いた場合も、演奏および音声出力が自動停止しますが、フォアグラウンド状態への復帰の際、SmartAudio16を動作させるコールバック機能は停止したままです。この点がHOMEボタンで停止した場合と異なります。

  コールバック機能を再開させるには、次のメソッドを呼び出します。

```
-(void)cktlResumeSystem;
```

- **AndroidOSの場合**

  AndroidOSでは明示的に演奏の停止処理を行わない限り、アプリケーションがバックグラウンドに移行してもSmartAudio16の音が鳴り続けてしまいます。
  SmartAudio16の演奏および音声出力を停止させるには、次のメソッドを呼び出します。

```
void cktlSuspendSystem(void);
```

  cktlSuspendSystemメソッドにより停止させた状態から、SmartAudio16の演奏および音声出力を再開させる場合には、次のメソッドを呼び出します。

```
void cktlResumeSystem(void);
```

9.その他のメソッドについて

CktlDriverクラスには、このガイドでとりあげたメソッド以外にも、例えばシーケンスの演奏状態を取得するためのメソッドや、サウンドヒープの使用状況を取得するためのメソッドなど、様々なメソッドが用意されています。

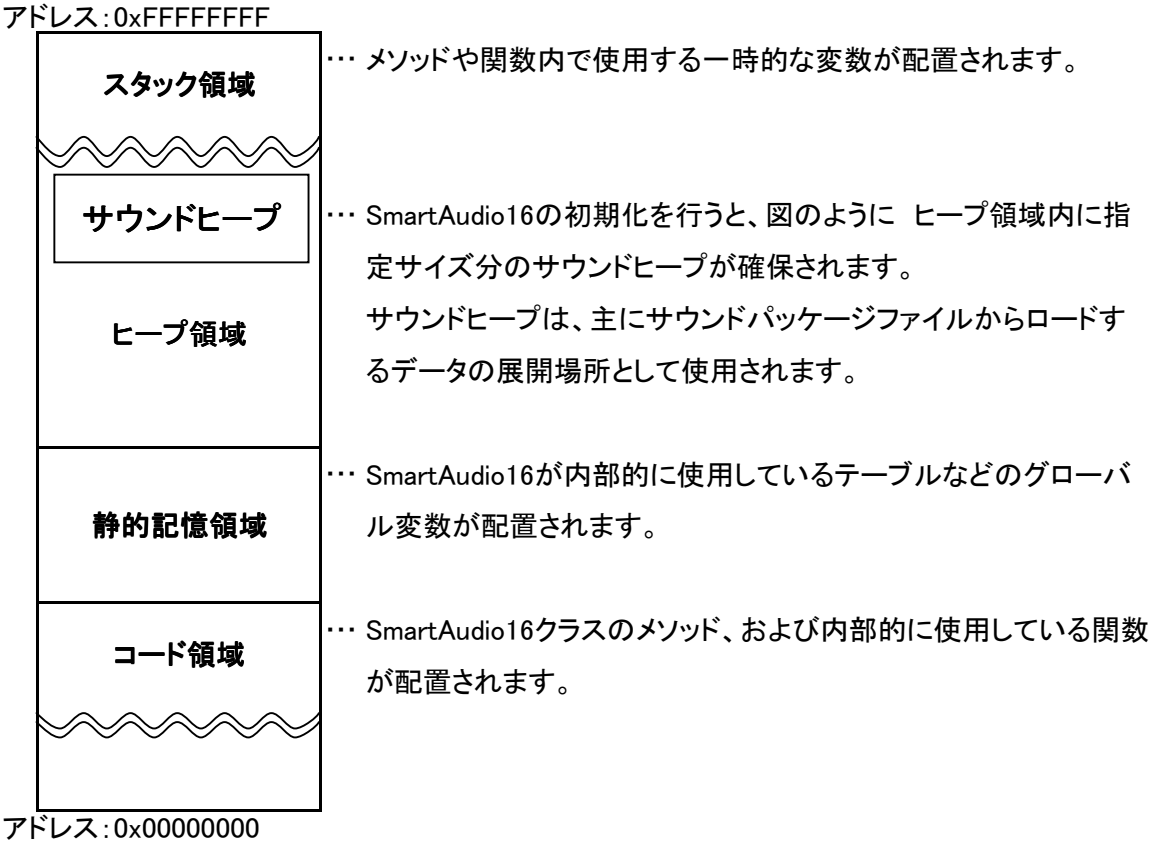
詳細は"SmartAudio16：メソッドリファレンス"ドキュメントを参照してください。

10.補足情報

SmartAudio16を使用する上での補足情報です。

10.1.実行時のメモリ展開

SmartAudio16のプログラムおよびそのプログラムが使用する変数、サウンドパッケージファイルに含まれるデータは、アプリケーション実行時に次の図のようにメモリへ配置されます。



メモリ展開のイメージ。

なお、関数や変数の定義はアプリケーションのものと混在して配置されるため、各領域の開始位置やサイズは、アプリケーションごとに異なります。

10.2.スレッドについて

SmartAudio16の初期化を行うと、オーディオコールバック用のスレッドを作成します。
オーディオコールバックとは、オーディオバッファを定期的に更新し音声を出力するための仕組みです。
オーディオバッファのサイズは各OS毎に定められています。
通常iOSでは約23msec、AndroidOSでは約58msecごとにコールバック用の関数が自動的に呼び出されます。
そのため、メインスレッド上でSmartAudio16に何らかの指示を与えてから音声出力に反映されるまで、最長で上記の時間分だけ遅延が発生する可能性があります。

改訂履歴

日付	内容
2014.02.26	ライブラリがバージョン1.4.0に変更されたことに伴い、『5.シーケンスの演奏』および『6.シーケンスの停止』の項を改訂。
2013.04.10	バージョン1.2.1に合わせて、『3.1.初期化時メモリサイズの目安』の項を改訂。
2013.03.01	SmartAudio16のバージョンが1.1.0に変更されたことに伴い、下記内容を変更。 レジデントインフォメーションのサウンドプレイヤーサイズを1816byteから1820byteに変更。
2013.02.02	ドキュメント改訂。
2012.08.04	新規ドキュメント作成。