

NITRO-Composer サウンドプログラマーガイド

2008-11-11

任天堂株式会社発行

このドキュメントの内容は、機密情報であるため、
厳重な取り扱い、管理を行ってください。

目次

1	はじめに	6
2	サウンドプログラム開発環境	6
2.1	TWL-Systemのビルド環境	6
2.2	ファイルの構成	6
2.2.1	ライブラリファイル	6
2.2.2	ヘッダファイル	6
2.2.3	サウンドデータ	6
2.2.4	ARM7 コンポーネント	7
3	サンプルツアー	8
3.1	開発環境	8
3.1.1	Makefile	8
3.1.2	ROM格納ファイル	8
3.1.3	動作手順	8
3.2	NitroMain関数	9
3.3	基本設定	10
3.3.1	OSの初期化など	10
3.3.2	サウンドライブラリの初期化	10
3.3.3	サウンドヒープの作成	10
3.3.4	サウンドアーカイブの初期化	10
3.3.5	プレイヤーのセットアップ	11
3.3.6	ストリームライブラリの初期化	11
3.3.7	サウンドフレームワーク	11
3.4	サウンドデータのロード	11
3.4.1	グループロード	11
3.5	シーケンス操作	12
3.5.1	サウンドハンドル	12
3.5.1.1	サウンドハンドルの使い方	12
3.5.1.2	サウンドハンドルとは？	12
3.5.1.3	シーケンスの切断	12
3.5.1.4	サウンドハンドル生成のヒント	12
3.5.2	シーケンスの再生	12
3.5.3	シーケンスアーカイブの再生	13
3.5.4	シーケンスの停止	13
3.6	その他のデモについて	13
3.6.1	stream	13
3.6.2	stream-2	13
3.6.3	stream-3	14
3.6.4	moveVolume	14
3.6.5	onMemory	14
3.6.6	reverb	14

3.6.7	effect	14
3.6.8	outputEffect	14
3.6.9	sampling	14
3.6.10	waveout	14
3.6.11	micThrough	14
3.6.12	driverInfo	15
4	ヒープ操作	16
4.1	概要	16
4.2	メモリ管理の基本	16
4.2.1	サウンドヒープとプレイヤーヒープ	16
4.2.2	2つのヒープの使い分け	16
4.3	サウンドヒープ操作	16
4.3.1	全てクリア	16
4.3.2	前の状態に戻す	17
4.3.3	複数のサウンドヒープ	17
4.4	プレイヤーヒープ操作	17
4.4.1	プレイヤーヒープの破棄	18
5	ストリーム再生	19
5.1	ストリームライブラリの初期化	19
5.1.1	ストリームスレッド	19
5.1.2	ストリームバッファ	19
5.2	ストリームの操作	19
5.2.1	ストリームハンドル	19
5.2.2	ストリームの再生	20
5.2.3	ストリームの停止	20
5.2.4	ストリームの一時停止	20
5.3	ストリームを途切れさせないために	20
5.3.1	ストリームスレッド	20
5.3.1.1	割り込み禁止	20
5.3.1.2	DMA	21
5.3.1.3	割り込みハンドラ処理	21
5.3.1.4	優先度の高いスレッド	21
5.3.2	カード・バックアップメディアのアクセス	21
5.3.3	ストリームバッファ	21
5.3.4	同時再生	21
6	注意点	22
6.1	スリープモード時のサウンド処理	22
6.1.1	復帰後の再生状態	22
6.1.1.1	シーケンス再生	22
6.1.1.2	ストリーム再生	22
6.1.1.3	サウンドキャプチャ	22
6.1.2	復帰後の安定動作待機	22

6.2	TWL-SDKのSND関数使用時の注意.....	22
6.2.1	プレイヤーの使用	23
6.2.2	チャンネルの使用	23
6.2.3	サウンドキャプチャの使用	23
6.2.4	サウンドアームの使用	23
6.2.5	SNDEX関数の使用	23
6.3	DSP使用時の注意.....	23
7	ライブラリ構成	24
7.1	ライブラリの構成	24
7.2	プレイヤーライブラリ.....	24
7.3	サウンドアーカイブプレイヤーライブラリ.....	25
7.4	サウンドアーカイブストリームライブラリ.....	25
7.5	ストリームライブラリ.....	26
7.6	サウンドアーカイブライブラリ.....	26
7.7	サウンドヒープライブラリ	26
7.8	キャプチャライブラリ	27
7.9	波形再生ライブラリ.....	27

コード

コード 3-1	Makefile.....	8
コード 3-2	NitroMain関数.....	9

表

表 7-1	プレイヤーライブラリ関数.....	25
表 7-2	サウンドアーカイブプレイヤーライブラリ関数	25
表 7-3	サウンドアーカイブストリームライブラリ関数	25
表 7-4	ストリームライブラリ関数	26
表 7-5	サウンドアーカイブライブラリ関数	26
表 7-6	サウンドヒープライブラリ関数.....	26
表 7-7	キャプチャライブラリ関数.....	27
表 7-8	波形再生ライブラリ関数	27

図

図 4-1	前の状態に戻す様子	17
図 7-1	ライブラリ構成図.....	24

改訂履歴

改訂日	改訂内容
2008-11-11	1. TWL-SDK の SND 関数使用時の注意を追加 2. DSP 使用時の注意を追加
2008-05-30	1. NITRO-System の名称変更による修正 (NITRO-System を TWL-System に変更)。
2008-04-08	1. 改訂履歴の書式を変更 2. ページのヘッダを修正
2006-05-29	1. スリープモード時のサウンド処理の説明修正 2. 誤記修正
2005-03-28	1. driverInfo デモの説明追加
2005-01-31	1. micThrough デモの説明追加 2. 波形再生ライブラリの説明補足 3. "NITRO"を"ニンテンドーDS"に変更
2004-12-06	1. stream-2.stream-3 デモの説明追加
2004-10-12	1. スリープモード移行時の説明追加 2. ストリームを途切れさせないための説明追加 3. sampling デモ、outputEffect デモの説明追加
2004-09-16	1. *.sadl ファイルの呼称を「サウンドラベルファイル」に統一
2004-09-02	1. サンプルソースコード変更に伴う修正
2004-08-10	1. ストリーム再生に関する説明追加 2. ストリームライブラリの説明追加
2004-07-20	1. 波形再生ライブラリ追加に伴う修正 2. エフェクト機能追加に伴う修正 3. 拡張子.bin を.srl に変更 4. ARM7 コンポーネントに関する説明修正
2004-06-01	1. ファイルシステム対応に伴う修正 2. 1つのプレイヤーで複数のシーケンスが再生できるようになったことに伴う修正 3. ヒープ操作の説明追加 4. ライブラリ構成の変更
2004-04-01	1. 全体的な構成変更 2. ライブラリ構成の説明追加 3. サンプルデモの概要説明追加
2004-03-18	1. SoundPlayer の Makefile 修正 2. OS_EnableIrqMask()の注意書き追加 3. テンポ変更関数の追加
2004-03-01	初版

1 はじめに

本ドキュメントでは、サウンドプログラマー向けに、サウンドプログラムを組むために最低限必要なことについて説明しています。

まず、サウンドプログラム開発環境の構築方法について説明し、続いてサンプルデモを見ながら、具体的な手順について説明します。最後の方では、サウンドライブラリの構成について説明し、どのような関数が用意されているのかを述べます。

個々の関数の詳細な説明については、関数リファレンスを参照してください。

2 サウンドプログラム開発環境

2.1 TWL-Systemのビルド環境

NITRO-Composer は、TWL-System の一部に入っています。そのため、TWL-System のビルド環境を構築するだけで、NITRO-Composer を扱えるようになります。

詳しくは、TWL-System のドキュメントを参照してください。

2.2 ファイルの構成

2.2.1 ライブラリファイル

ライブラリファイルは、2つあり TWL-SDK のものと、TWL-System のもの、両方をリンクする必要があります。

```
libsnd.a  
libnnsnd.a
```

2.2.2 ヘッドファイル

必要な関数用のヘッドファイルのインクルードは、

```
#include <nnsys/snd.h>
```

で、行えます。

また、サウンドデザイナー作成のサウンドラベルファイル(*.sdl)をインクルードすることで、サウンドデータの指定を番号ではなく、サウンドデザイナーが定義したラベルで行えるようになります。

```
#include "../data/sound_data.sdl"
```

2.2.3 サウンドデータ

サウンドデータは、拡張子*.sdat のサウンドアーカイブという1つのファイルに全てまとめられています。このサウンドアーカイブを ROM に格納するように設定します。

具体的な方法は、後のサンプルで説明します。

2.2.4 ARM7 コンポーネント

ARM7 コンポーネントは、TWL-SDK に格納されています。ARM7 コンポーネントには、サウンド機能が実装されている必要がありますが、明示的に ARM7 コンポーネントを指定していない場合は、サウンド機能が実装されたコンポーネントが使用されるようになっています。

3 サンプルツアー

単純な構成のサンプルデモ「simple」を使いながら説明します。「simple」は、\$TwlSystem/build/demo/snd/simple にあります。

3.1 開発環境

開発環境の構築方法について、説明します。

3.1.1 Makefile

まずは下記の Makefile を見てください。ただし、コメントなど一部省略しています。

コード 3-1 Makefile

```
#!/ make -f

#-----

SRCS          =      main.c

TARGET_NEF    =      main.nef
TARGET_BIN    =      main.srl

MAKEROM_ROMROOT = ../data
MAKEROM_ROMFILES = sound_data.sdat

include      $(TWLSYSTEM_ROOT)/build/buildtools/commondefs

#-----

do-build:     $(TARGETS)

include      $(TWLSYSTEM_ROOT)/build/buildtools/modulerrules

#==== End of Makefile ====
```

基本的な箇所については、説明を省略します。TWL-SDK や TWL-System のマニュアルを参照してください。

重要なのは、MAKEROM_で始まる 2 つの変数の設定です。

3.1.2 ROM格納ファイル

MAKEROM_ROMROOT で ROM に格納するファイルのルートディレクトリを、MAKEROM_ROMFILES で ROM に格納するファイルを、それぞれ指定しています。すなわちこの2つで、"./data/sound_data.sdat"というパスのファイルが、ROM に格納されることになります。

3.1.3 動作手順

上記の Makefile ファイルを使うと、次のような手順で実行されます。

- SRCS に登録されている main.c をコンパイルします。
- コンパイルでできたものとライブラリをリンクして、ARM9 コンポーネント main ができます。
- ARM9 コンポーネント main と、ARM7 コンポーネント及びサウンドアーカイブをまとめて、main.srl を作ります。
- main.srl が実行ファイルになります。

3.2 NitroMain関数

まずは、src/main.c 中 NitroMain 関数を見てください。ただし、コメントなど一部省略しています。

コード 3-2 NitroMain 関数

```
void NitroMain()
{
    OS_Init();
    GX_Init();

    // VBlank settings
    OS_SetIrqFunction(OS_IE_V_BLANK, VBlankIntr);
    (void)OS_EnableIrqMask( OS_IE_V_BLANK );
    (void)OS_EnableIrq();
    (void)GX_VBlankIntr(TRUE);

    FS_Init( MI_DMA_MAX_NUM );

    // Initialize sound
    NNS_SndInit();
    heap = NNS_SndHeapCreate( & sndHeap, sizeof( sndHeap ) );
    NNS_SndArcInit( &arc, "/sound_data.sdat", heap, FALSE );
    (void)NNS_SndArcPlayerSetup( heap );
    NNS_SndArcStrmInit( STREAM_THREAD_PRIO, heap );

    // Load sound data
    (void)NNS_SndArcLoadSeq( SEQ_MARIOKART64_TITLE, heap );
    (void)NNS_SndArcLoadSeqArc( SEQ_SE, heap );
    (void)NNS_SndArcLoadBank( BANK_SE, heap );

    // Initialize sound handle
    NNS_SndHandleInit( &bgmHandle );
    NNS_SndHandleInit( &seHandle );

    // dummy pad read
    Cont = PAD_Read();

    //===== Main Loop
    while(1)
    {
        u16 ReadData;

        SVC_WaitVBlankIntr();

        ReadData = PAD_Read();
        Trg = (u16)(ReadData & (ReadData ^ Cont));
        Cont = ReadData;

        if ( Trg & PAD_BUTTON_A ) {
            // start BGM
            (void)NNS_SndArcPlayerStartSeq( &bgmHandle, SEQ_MARIOKART64_TITLE
);
        }
    }
}
```

```
    if ( Trg & PAD_BUTTON_B ) {  
        // stop BGM  
        (void)NNS_SndPlayerStopSeq( &bgmHandle, 1 );  
    }  
  
    if ( Trg & PAD_KEY_UP ) {  
        // start SE  
        (void)NNS_SndArcPlayerStartSeqArc( &seHandle, SEQ_SE, SE_COIN );  
    }  
  
    //---- framework  
    NNS_SndMain();  
}  
}
```

以下、ポイントとなる点について、順番に説明していきます。

3.3 基本設定

まずは、ライブラリの初期化など、必要最低限な関数について説明します。

3.3.1 OSの初期化など

まずは、必要に応じて OS の初期化などを行います。

```
OS_Init();  
GX_Init();  
  
// VBlank settings  
OS_SetIrqFunction(OS_IE_V_BLANK, VBlankIntr);  
(void)OS_EnableIrqMask( OS_IE_V_BLANK );  
(void)OS_EnableIrq();  
(void)GX_VBlankIntr(TRUE);  
  
FS_Init( MI_DMA_MAX_NUM );
```

3.3.2 サウンドライブラリの初期化

サウンドライブラリの初期化を行います。全ての NNS_Snd 関数より前に呼び出す必要があります。

```
NNS_SndInit();
```

3.3.3 サウンドヒープの作成

サウンドデータを格納するためのヒープを作成します。

```
heap = NNS_SndHeapCreate( & sndHeap, sizeof( sndHeap ) );
```

1つ目の引数はヒープとして使用するメモリの先頭アドレスで、2つの目の引数はメモリのサイズです。

返り値としてヒープハンドルが返ります。このサウンドヒープからメモリを確保するために、ヒープハンドルを使います。

3.3.4 サウンドアーカイブの初期化

サウンドアーカイブを初期化します。なお、サウンドアーカイブ構造体は、静的に確保しなければなりません。

```
NNS_SndArcInit( &arc, "/sound_data.sdat", heap, FALSE );
```

1 つ目の引数がサウンドアーカイブ構造体、2 つ目の引数が ROM ファイルシステム上でのサウンドアーカイブのパスです。

3 つ目の引数はサウンドアーカイブの初期化に必要なメモリを確保するためのヒープで、先ほど作成したサウンドヒープのハンドルを入れます。ここで確保したメモリを解放してしまうと、サウンドアーカイブが使えなくなってしまうので、注意してください。

4 つ目の引数は、サウンドアーカイブ中のシンボルデータをロードするかどうかのフラグです。シンボルデータはデバッグなどの目的で使いますが、通常は不要ですので FALSE を入れてください。

3.3.5 プレイヤーのセットアップ

プレイヤーをセットアップをします。

```
NNS_SndArcPlayerSetup( heap );
```

サウンドアーカイブ中で定義されているプレイヤー設定に基づき、プレイヤーをセットアップします。

プレイヤーのセットアップには、メモリが必要な時がありますので、引数にヒープハンドルを入れます。

3.3.6 ストリームライブラリの初期化

ストリーム再生を行う場合は、ストリームライブラリを初期化する必要があります。

```
NNS_SndArcStrmInit( STREAM_THREAD_PRIO, heap );
```

ストリームに関して詳しくは、「5章 ストリーム再生」で説明します。

3.3.7 サウンドフレームワーク

サウンドライブラリのフレームワークを行います。1 フレームに1度呼び出しさえすれば、どこでコールしても構いません。

```
NNS_SndMain();
```

3.4 サウンドデータのロード

シーケンスを再生する前に、必要なデータをロードしておく必要があります。

```
(void)NNS_SndArcLoadSeq( SEQ_MARIOKART64_TITLE, heap );  
(void)NNS_SndArcLoadSeqArc( SEQ_SE, heap );  
(void)NNS_SndArcLoadBank( BANK_SE, heap );
```

NNS_SndArcLoadSeq 関数で、シーケンス SEQ_MARIOKART64_TITLE を再生するために必要なデータをロードします。これは、シーケンスデータだけでなく、バンクデータや波形データも同時にロードします。

NNS_SndArcLoadSeqArc 関数は、SE のシーケンスアーカイブをロードします。シーケンスアーカイブは、複数のバンクと関連があるため、自動的にバンクデータや波形データはロードされません。次の NNS_SndArcLoadBank 関数で SE 用のバンクデータをロードしています。この関数ではバンクデータだけでなく、必要な波形データもロードされますので、波形データを別にロードする必要はありません。

3.4.1 グループロード

実際には、サンプルのようにデータを1つ1つ指定してロードすることは、あまりありません。サウンドデザイナーがグループを定義していれば、次のようにして簡単にロードすることができます。

```
(void)NNS_SndArcLoadGroup( GROUP_STATIC, heap );
```

グループには、どのデータをロードするかが定義してあります。NNS_SndArcLoadGroup 関数を呼ぶと、それら全てのデータが一度にロードされます。

グループロードを使うことで、プログラムを変更せずに、ロードするデータを修正することができるようになります。

3.5 シーケンス操作

3.5.1 サウンドハンドル

3.5.1.1 サウンドハンドルの使い方

シーケンスを扱うためには、サウンドハンドルというものがようになります。

```
NNSSndHandle bgmHandle;  
NNSSndHandle seHandle;
```

サウンドハンドルは、とりあえず静的に確保しておきます。また、使い始める前には、必ず次の関数で初期化しておきます。

```
void NNS_SndHandleInit( NNSSndHandle* handle );
```

3.5.1.2 サウンドハンドルとは？

サウンドハンドルは、シーケンスを再生後も制御するためのオブジェクトです。1つのサウンドハンドルで1つのシーケンスの制御ができます。シーケンスの再生に成功すると、サウンドハンドルにシーケンスが結びつけられます。以降、この結びつきが切断されるまで、サウンドハンドルに対する操作は、そのままシーケンスへの操作となります。

3.5.1.3 シーケンスの切断

シーケンスの切断は明示的に行う場合と、不意に起こる場合があります。不意に起こるのは、例えば、同時に1つのシーケンスしか再生できないプレイヤーで別のシーケンスをスタートさせたときに、再生中のシーケンスが強制的に止められたときなどです。この様なとき、サウンドハンドルは自動的にシーケンスと切断させられ、無効化されます。無効化したサウンドハンドルに対して操作を行っても、何も処理されません。

以上のことは、プログラマーが再生したシーケンスが、本当にまだ再生中なのかを意識しなくてすむ事を意味します。再生中の時と、すでに停止していた時とで、同じ処理を行っても、誤って別のシーケンスの操作を行ってしまうという問題は発生しません。

3.5.1.4 サウンドハンドル生成のヒント

ワンショットのSEなど、再生だけして停止などは行わない場合には、サウンドハンドルを1つだけ用意して、使い回して次々に再生することができます。シーケンス最大同時再生数の制限に引っかからなければ、全ての音は同時に再生されます。また、再生直後だけなら、それぞれに対してパラメータの変更も行うことができます。

BGM やエンジン音などの持続系のSEでは、すくなくとも停止する必要がありますので、それぞれに対してサウンドハンドルが必要になります。

3.5.2 シーケンスの再生

シーケンスを再生する関数は、次のものです。

```
BOOL NNS_SndArcPlayerStartSeq( NNSSndHandle* handle, int seqNo );
```

seqNo はシーケンス番号で、サウンドアーカイブでの並び順になります。

関数が成功すると、引数に渡したサウンドハンドルにシーケンスが結びつけられます。以後、このサウンドハンドルを用いて、シーケンスの停止などが行えます。

もし、サウンドハンドルが、すでにシーケンスに結びつけられていた場合は、元のシーケンスとの結合は切断され、新しいシーケンスと結合されます。サウンドハンドルと切り離されたシーケンスは、直接制御することができなくなりますので、注意してください。ただし、ワンショットのSEを鳴らすときなど、後でシーケンスを制御する必要のない場合には、特に注意する必要はありません。

```
if ( Trg & PAD_BUTTON_A ) {  
    (void)NNS_SndArcPlayerStartSeq(&bgmHandle, SEQ_MARIOKART64_TITLE );  
}
```

3.5.3 シーケンスアーカイブの再生

シーケンスアーカイブのシーケンスを再生する関数は、次のものです。

```
BOOL NNS_SndArcPlayerStartSeqArc(  
    NNSSndHandle* handle, int seqArcNo, int index );
```

seqArcNo はシーケンスアーカイブの番号で、サウンドアーカイブでの並び順になります。index は、シーケンスのインデックス番号で、シーケンスアーカイブでの並び順になります。そのほかは、シーケンスの再生と同じです。

```
if ( Trg & PAD_KEY_UP ) {  
    (void)NNS_SndArcPlayerStartSeqArc( &seHandle, SEQ_SE, SE_COIN );  
}
```

3.5.4 シーケンスの停止

シーケンスを停止する関数は、次のものです。

```
void NNS_SndPlayerStopSeq( NNSSndHandle* handle, int fadeFrame );
```

handle は、再生したときに渡したサウンドハンドルを入れます。

fadeFrame は、フェードアウトフレームです。指定したフレーム数かけて、徐々に音量を落としていきます。

```
if ( Trg & PAD_BUTTON_B ) {  
    (void)NNS_SndPlayerStopSeq( &bgmHandle, 1 );  
}
```

3.6 その他のデモについて

「simple」デモで使用されている関数についての説明は以上ですが、その他のデモについても、概要を説明しておきます。なお、NITRO-Composer のデモプログラムは全て、\$TwlSystem/build/demo/snd 以下に格納されています。

3.6.1 stream

ストリーム再生を行うデモです。ストリーム再生については、「5章 ストリーム再生」でも説明します。

3.6.2 stream-2

複数のストリームデータをリアルタイムに結合して再生を行うデモです。NNS_SndArcStrmStartEx2 関数を使って、結合のための処理を行うコールバック関数を登録しています。

3.6.3 stream-3

ストリームにエフェクトをかけて再生を行うデモです。NNS_SndArcStrmStartEx2 関数を使って、エフェクト処理を行うコールバック関数を登録しています。

3.6.4 moveVolume

シーケンス全体のボリュームを時間変化させるデモです。NNS_SndPlayerMoveVolume 関数を使って、ボリュームの変更を行っています。フェードイン再生のコードもあります。

3.6.5 onMemory

サウンドアーカイブを全てメモリ上に置いて使用するデモです。そのために、NNS_SndArcInitOnMemory 関数を使って、サウンドアーカイブを初期化しています。

3.6.6 reverb

キャプチャ機能を使ったリバーブのデモです。

NNS_SndCaptureStartReverb 関数や NNS_SndCaptureStopReverb 関数などを使っています。

3.6.7 effect

キャプチャ機能を使ったエフェクトのデモです。簡単なローパスフィルタ(移動平均)をかけて出力します。

NNS_SndCaptureStartEffect 関数や NNS_SndCaptureStopEffect 関数などを使っています。

3.6.8 outputEffect

キャプチャ機能を使ったエフェクトのデモです。サラウンドモードやヘッドフォンモードに切り替えて出力します。

NNS_SndCaptureStartOutputEffect 関数や NNS_SndCaptureChangeOutputEffect 関数などを使っています。

3.6.9 sampling

キャプチャ機能を使ったサンプリングのデモです。サンプリングデータを元に出力レベルを計算して表示しています。

NNS_SndCaptureStartSampling 関数などを使っています。

3.6.10 waveout

シーケンス再生ではなく、波形データを直接再生するデモです。マイクで録音した音を、再生しています。波形データの再生には、NNS_SndWaveOutStart 関数を使っています。

3.6.11 micThrough

低レベルストリームライブラリ NNS_SndStrm を使ったデモです。マイク入力をリアルタイムに再生しています。また、出力音声にエフェクトをかけることもできます。

NNS_SndStrmSetup 関数や NNS_SndStrmStart 関数などを使っています。

3.6.12 driverInfo

サウンドドライバの情報を画面上に表示するデモです。

NNS_SndUpdateDriverInfo 関数でサウンドドライバの情報を更新し、NNS_SndPlayerReadDriverPlayerInfo 関数などで、サウンドドライバにおけるプレイヤーの情報を取得しています。

4 ヒープ操作

4.1 概要

「simple」デモでは、メモリ管理についてほとんど触れられていません。初期化時に、`NNS_SndHeapCreate` 関数でサウンドヒープを作成し、後は必要に応じてデータをロードしただけでした。

ここでは、ヒープ操作の方法について説明していきます。

4.2 メモリ管理の基本

メモリ管理の基本について詳しくは、「サウンドシステムマニュアル」を参照してください。ここでは、簡単に説明します。

4.2.1 サウンドヒープとプレイヤーヒープ

ヒープは、サウンドヒープとプレイヤーヒープの2種類用意されています。

サウンドヒープはスタック方式のヒープで、プログラマーがデータのロードや削除などの管理を行います。

一方プレイヤーヒープは、シーケンス再生時に自動的にデータをロードして使用されるヒープで、プログラマーが直接操作する必要はありません。

4.2.2 2つのヒープの使い分け

サウンドヒープには、起動時や場面切り替え時などに、比較的大きなデータをロードします。プレイヤーヒープには、BGMシーケンスデータなど、比較的小さいデータが、シーケンスの再生時にロードされます。

ただしこれは一般的な使い方であり、ロード効率重視で全てサウンドヒープだけで管理することなどもできます。

4.3 サウンドヒープ操作

サウンドヒープはスタック方式のため、上から順番に確保され、解放は下から順に行います。ヒープからのメモリの確保は、サウンドデータをロードするときに自動的に行われます。一方、不要になったサウンドデータを削除するには、メモリ領域を解放します。メモリ領域の解放は、次の2つの方法で行います。

- 全てクリア
- 前の状態に戻す

4.3.1 全てクリア

単純に全部のサウンドデータを消して、一番最初の状態に戻すことができます。非常に単純ですが、これを行うと、再生中の全ての音が止まってしまいます。また、サウンドアーカイブの初期化時に使ったメモリ領域を解放してしまうと、サウンドアーカイブが使えなくなってしまうので、注意しなければなりません。

全てクリアするためには、`NNS_SndHeapClear` 関数を呼ぶだけです。

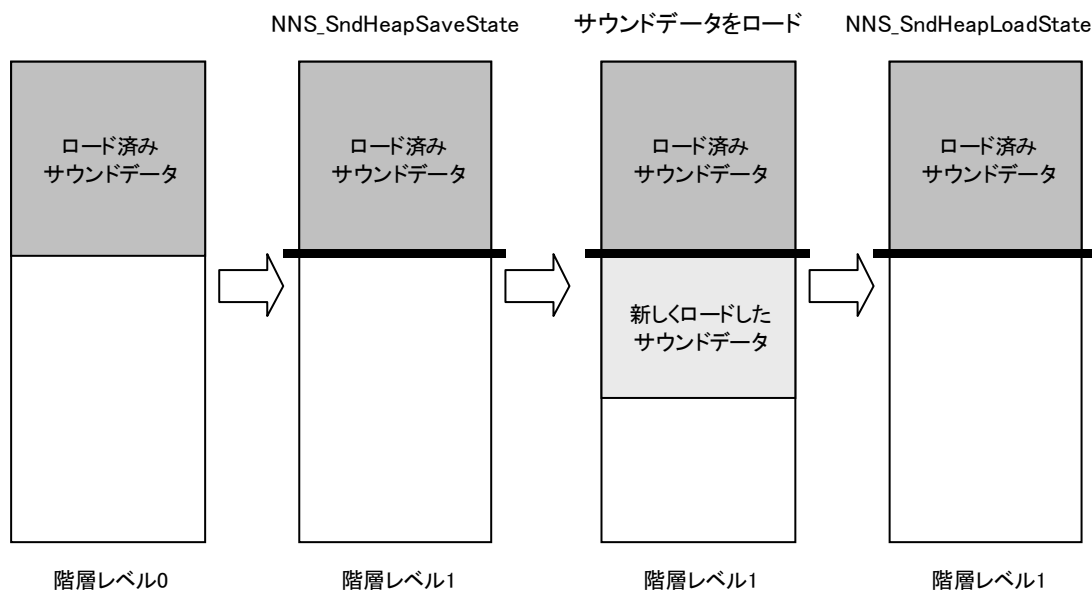
4.3.2 前の状態に戻す

通常はこちらの方法を使います。

まずは、NNS_SndHeapSaveState 関数で、現在の状態を保存します。返回值として、保存後の階層レベルが返ります。階層レベルとは、保存した状態を指し示す値で、この値を使って、サウンドヒープを保存した状態に戻すことができます。

幾つかのサウンドデータをロードした後、NNS_SndHeapLoadState 関数に先ほどの階層レベルの値を入れて呼ぶと、NNS_SndHeapSaveState 関数を呼んだ直後の状態に戻すことができます。すなわち、NNS_SndHeapSaveState 関数を呼んだ後にロードしたデータが削除されます。

図 4-1 前の状態に戻す様子



この時、ロード済みサウンドデータを使って再生されている音は止まりません。

また、NNS_SndHeapSaveState 関数は繰り返し呼ぶことができ、呼ぶ毎に階層レベルの値は大きくなります。

4.3.3 複数のサウンドヒープ

通常サウンドヒープは1つだけ用意しますが、複数作成することもできます。複数用意することで、それぞれ独立した状態の保存及び復帰が行えます。

複数のサウンドヒープを使うためには、NNS_SndHeapCreate 関数で複数のサウンドヒープを作成するだけです。サウンドヒープからメモリを確保するときは、必ずヒープハンドルを指定する必要がありますので、返回值のヒープハンドルを関数の引数に入れて、どのサウンドヒープからメモリを確保するのかを指定します。

4.4 プレイヤーヒープ操作

通常、プログラマーがプレイヤーヒープを管理する必要はありません。プレイヤーヒープは、次の関数内で作成されます。

```
BOOL NNS_SndArcPlayerSetup( NNSndHeapHandle heap );
```

どのプレイヤーに対し、どのサイズのヒープを作成するのかといった情報は、サウンドデザイナーがサウンドアーカイブで設定しています。引数で渡したサウンドヒープから適当にメモリが確保され、プレイヤーヒープが作成されます。

4.4.1 プレイヤーヒープの破棄

プレイヤーヒープは、サウンドヒープからメモリを確保します。このプレイヤーヒープを確保したメモリ領域を解放すると、プレイヤーヒープは自動的に破棄されます。

5 ストリーム再生

ここでは、「simple」デモには含まれていなかったストリーム再生について、説明します。

5.1 ストリームライブラリの初期化

ストリーム再生を行う場合は、ストリームライブラリを初期化する必要があります。

```
NNS_SndArcStrmInit( STREAM_THREAD_PRIO, heap );
```

1つ目の引数は、ストリームスレッドのスレッドプライオリティです。2つ目の引数は、ストリームバッファを確保するためのサウンドヒープハンドルです。

5.1.1 ストリームスレッド

ストリームスレッドは、ストリームデータを ROM からロードする必要があるときに、データのロード処理などを行うスレッドです。

ストリーム再生では、再生しながらデータのロードを行うため、データのロードが再生に間に合わないと、音が途切れてしまいます。そのため、データをロードする必要があるとき、速やかにロードを実行する必要があります。ストリームスレッドは、例えばゲームのメインループで別の処理をしている途中でであっても、データロードの必要があれば、その処理に割り込んで先にロード処理を行わせることができます。

ストリームスレッドが、メインループ処理に割り込んで処理できるようにするためには、ストリームスレッドのスレッドプライオリティを高く設定しておく必要があります。通常メインループ(メインスレッド)のプライオリティは 16 なので、それより小さい値を指定します。

5.1.2 ストリームバッファ

ストリーム再生のためには、データをロードするためのバッファが必要です。1つのストリームを再生するのに、2~4KB 程度のバッファが必要です。このバッファは、2つ目の引数に渡したサウンドヒープから取得されます。

このストリームバッファを確保したメモリ領域を解放してしまうと、ストリームを再生できなくなってしまうので、注意してください。

5.2 ストリームの操作

5.2.1 ストリームハンドル

シーケンス再生のために、サウンドハンドルが必要だったように、ストリームの再生にはストリームハンドルが必要になります。

```
NNSSndStrmHandle strmHandle;
```

ストリームハンドルは、とりあえず静的に確保しておきます。また、使い始める前には、必ず次の関数で初期化しておきます。

```
void NNS_SndStrmHandleInit( NNS_SndStrmHandle* handle );
```

その他の使い方に関しても、サウンドハンドルと全く同じです。

5.2.2 ストリームの再生

ストリームの再生を行うためには、次の関数を呼び出します。

```
BOOL NNS_SndArcStrmStart( NNSndStrmHandle* handle, int strmNo, u32 offset );
```

`strmNo` はストリーム番号です。どのストリームを再生するのかを指定します。`offset` は、ストリームデータ中の再生開始位置を msec 単位で指定します。通常は始めから再生するので、0 を入れます。

再生に成功すると、ストリームハンドルにストリームが結びつけられるのも、シーケンスの時と同じです。

5.2.3 ストリームの停止

ストリームの停止を行うためには、次の関数を呼び出します。

```
void NNS_SndArcStrmStop( NNSndStrmHandle* handle, int fadeFrames );
```

`fadeFrames` で指定したフレーム数かけて、徐々に音量を落とした後完全に停止します。`fadeFrames` に 0 を入れると、即座に停止します。

5.2.4 ストリームの一時停止

ストリームを一時停止できる関数は用意されていません。ただし、一時停止に似た処理は可能です。方法は以下の通りです。

まず、一時停止したくなったら、次の関数で現在の再生位置を取得します。

```
u32 NNS_SndArcStrmGetCurrentPlayingPos( NNSndStrmHandle* handle );
```

その後、`NNS_SndArcStrmStop` 関数でストリームを停止させます。

次にストリーム再生を再開したくなったら、先ほど取得した再生位置を、`NNS_SndArcStrmStart` 関数の `offset` 引数に入れて、ストリーム再生を開始します。すると、停止した位置から再生を始めることができます。

ただし、厳密に同じ位置からの再生再開ではないことに注意してください。

5.3 ストリームを途切れさせないために

ストリーム再生では、リアルタイムにデータのロードを行います。ロードが間に合わないと音が途切れてしまいます。以下、音を途切れさせないようにするためのヒントです。

5.3.1 ストリームスレッド

ストリームデータのロードは、ストリームスレッドで行います。つまり、ストリームスレッドの処理を遅れなく動かすことが、ストリームを途切れさせないことの基本です。

ストリームスレッドの処理を遅らせる要因となるものは、以下のものです。

5.3.1.1 割り込み禁止

割り込み禁止中は、ストリームスレッドが動くことができません。割り込み禁止区間は、できるだけ短くする必要があります。

5.3.1.2 DMA

DMA 駆動中はストリームスレッドが動くことはできません。大きな DMA は分割することで、ストリームスレッド処理の遅れを緩和することができます。

5.3.1.3 割り込みハンドラ処理

割り込みハンドラで処理中は、ストリームスレッドが動くことはできません。割り込みハンドラ処理は、できるだけ短くする必要があります。

5.3.1.4 優先度の高いスレッド

ストリームスレッドより優先度の高いスレッドの処理中は、ストリームスレッドが動くことはできません。優先度の高いスレッドの処理を短くするか、ストリームスレッドの優先度を上げる必要があります。

5.3.2 カード・バックアップメディアのアクセス

カードやバックアップメディアにアクセス中は、ストリームデータをロードできませんので、アクセスを分割するなどの対処が必要です。

5.3.3 ストリームバッファ

ストリーム再生用のバッファが大きいと、多少ストリームスレッドの処理が遅れても、音が途切れないようになります。ただし、バッファを大きくすると、ストリームスレッド1回当たりの処理が大きくなり、ストリームスレッドより優先度の低いスレッドに悪影響を及ぼす場合があります。

5.3.4 同時再生

ストリームを複数同時に再生すると、ストリームスレッドの処理が大きくなり、少しの遅延でも音が途切れる場合があります。複数同時に再生するときは、特に注意が必要です。

6 注意点

6.1 スリープモード時のサウンド処理

スリープモード移行時に、サウンド機能を停止させる処理はライブラリが自動で行います。すなわち、プログラマーがスリープモード移行関数 `PM_GoSleepMode` を呼び出すだけで、必要な処理はライブラリ側で実行されます。

また、スリープモードからの復帰時に、サウンド機能を再開させる処理も、ライブラリ側で行います。ただし、下記の点には注意してください。

6.1.1 復帰後の再生状態

スリープモード復帰後の再生状態は、スリープ前の再生状態と完全に一致するわけではありません。次のような問題が発生します。

6.1.1.1 シーケンス再生

シーケンス処理自体は、スリープモード移行時の位置から再開されますが、シーケンスで発音中のチャンネルは、スリープモード復帰時に波形データの一番始めから再生されます。

これが大きな問題になる場合は、プログラマ側でスリープ前に再生を一時停止し、スリープ後に一時停止を解除するといった対処が必要になります。ただし、発音中のチャンネルは、スリープモード復帰時に再生されません。

6.1.1.2 ストリーム再生

スリープ前に再生を停止し、スリープ後に再生を再開する処理を行っています。この時、ストリームバッファに溜まっていたデータは破棄されるため、音が部分的に飛んでしまいます。

この問題への有効な対処方法はありません。

6.1.1.3 サウンドキャプチャ

スリープ前にキャプチャ機能を停止し、スリープ後に機能を再開する処理を行っています。この時、キャプチャバッファに溜まっていたデータは破棄されるため、スリープ復帰直後のキャプチャデータは 0 になります。

この問題への有効な対処方法はありません。

6.1.2 復帰後の安定動作待機

スリープモードから復帰した後、サウンド回路が安定動作するまで 15msec の時間がかかりますが、この待機時間は、ライブラリ関数 (`PM_GoSleepMode` 関数) 内で処理されています。従って、プログラマがこの点に留意する必要はありません。

6.2 TWL-SDKのSND関数使用時の注意

NITRO-Composer は、TWL-SDK の SND 関数を使用して実装されています。そのため、TWL-SDK の SND 関数を直接使う場合は、以下の点に注意する必要があります。

6.2.1 プレイヤーの使用

NITRO-Composer では、SND_StartSeq などのシーケンスコマンド関数を使用してシーケンスを再生しています。

NITRO-Composer 使用中に、シーケンスコマンド関数を呼び出すことは禁止です。

6.2.2 チャンネルの使用

NITRO-Composer の NNS_SndLockChannel 関数を呼び出すと、指定したチャンネルは、NITRO-Composer から使用されなくなります。このチャンネルに対してのみ、SND 関数を使ってチャンネル操作を行うことができます。

6.2.3 サウンドキャプチャの使用

NITRO-Composer の NNS_SndLockCapture 関数を呼び出すと、指定したサウンドキャプチャは、NITRO-Composer から使用されなくなります。このサウンドキャプチャに対してのみ、SND 関数を使ってサウンドキャプチャ操作を行うことができます。

ただし、サウンドキャプチャは、チャンネル1またはチャンネル3も使用するため、同時に NNS_SndLockChannel 関数も呼び出す必要があることに注意してください。

6.2.4 サウンドアラームの使用

NITRO-Composer は内部でサウンドアラームを使用しますので、NITRO-Composer を使用している場合にサウンドアラームを使うには、NITRO-Composer の NNS_SndAllocAlarm 関数を呼び出して、使用可能なアラーム番号を取得してください。取得した番号のアラームは、NITRO-Composer 内部では使用しません。

使用が終わったアラームは NNS_SndFreeAlarm で解放してください。

6.2.5 SNDEX関数の使用

SNDEX 関数は、NITRO-Composer と独立して使用することができます。

ただし、SNDEX_SetVolume 関数など、NITRO-Composer の音声出力にも影響を与える関数があることに注意してください。

6.3 DSP使用時の注意

DSP は、NITRO-Composer と独立して使用することができます。

ただし、SNDEX_SetDSPMixRate 関数で、DSP の出力音量を上げると、NITRO-Composer からの出力音量が下がることに注意してください。

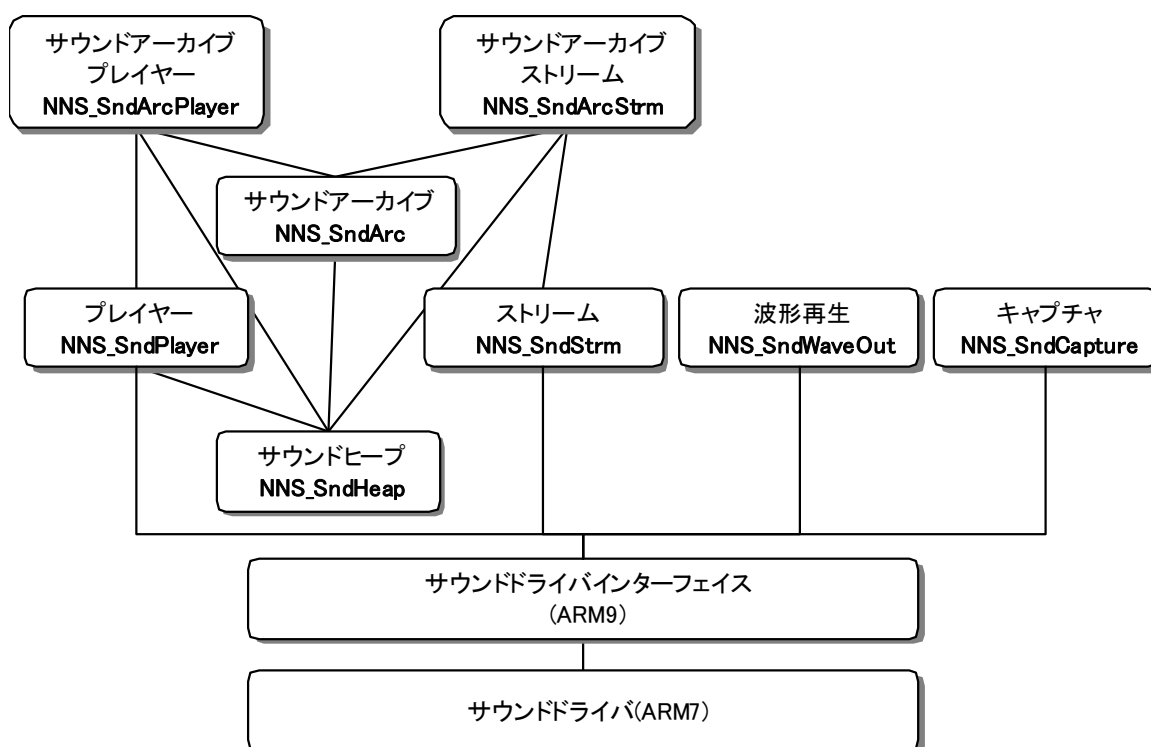
7 ライブラリ構成

NITRO-Composer 内部のライブラリ構成について説明します。

7.1 ライブラリの構成

NITRO-Composer ライブラリは、内部的に下図の様な幾つかのライブラリで構成されています。

図 7-1 ライブラリ構成図



NNS_Snd で始まる接頭辞がついたライブラリが TWL-System に含まれるサウンドライブラリで、プログラマーは通常これらのライブラリ関数を使用します。以下、個々のライブラリについて説明していきます。

7.2 プレイヤーライブラリ

シーケンスを再生するための一番基本となるライブラリです。関数名は、NNS_SndPlayer で始まります。

シーケンスのパラメータ変更や、シーケンスの停止などは、このライブラリを使って行いますが、シーケンスの再生だけに関しては、上位ライブラリのサウンドアーカイブプレイヤーの関数を使います。プレイヤーライブラリのシーケンス再生関数は、単純にシーケンスを走らせるだけの処理を行いますが、実際にはシーケンスの再生に当たっては、データのロードなどの複雑な処理が必要になります。このため、再生の部分はサウンドアーカイブプレイヤーで処理するようになっています。

また、プレイヤーライブラリには、一部サウンドハンドルを扱う関数も含まれています。これらの関数名は、NNS_SndHandle で始まります。サウンドハンドルと、プレイヤーは密接にかかわっています。サウンドハンドルの概要については、「

3.5.1サウンドハンドル」を参照してください。

プレイヤーライブラリの主な関数は以下の通りです。

表 7-1 プレイヤーライブラリ関数

関数名	説明
NNS_SndPlayerStopSeq	シーケンスを停止します。
NNS_SndPlayerPause	シーケンスを一時停止または再開します。
NNS_SndPlayerSetTempoRatio	シーケンスのテンポを変更します。
NNS_SndPlayerSetVolume	シーケンスのボリュームを変更します。
NNS_SndPlayerSetTrackVolume	シーケンストラックのボリュームを変更します。
NNS_SndPlayerSetTrackPitch	シーケンストラックの音程を変更します。
NNS_SndPlayerSetTrackPan	シーケンストラックのパン(定位)を変更します。

7.3 サウンドアーカイブプレイヤーライブラリ

サウンドアーカイブを用いて、シーケンスを再生するライブラリです。関数名は、NNS_SndArcPlayer で始まります。

プレイヤーライブラリとサウンドアーカイブライブラリの上位ライブラリに位置し、それぞれの機能を使って、簡単にシーケンスを再生できるようにします。

サウンドアーカイブプレイヤーライブラリの主な関数は、以下の通りです。

表 7-2 サウンドアーカイブプレイヤーライブラリ関数

関数名	説明
NNS_SndArcPlayerSetup	サウンドアーカイブの設定に基づいて、プレイヤーをセットアップします。
NNS_SndArcPlayerStartSeq	シーケンスを再生します。
NNS_SndArcPlayerStartSeqArc	シーケンスアーカイブを再生します。

7.4 サウンドアーカイブストリームライブラリ

サウンドアーカイブ中のストリームデータを再生するライブラリです。関数名は、NNS_SndArcStrm で始まります。

ストリームライブラリとサウンドアーカイブライブラリの上位ライブラリに位置し、それぞれの機能を使って、簡単にストリームを再生できるようにします。

サウンドアーカイブストリームライブラリの主な関数は、以下の通りです。

表 7-3 サウンドアーカイブストリームライブラリ関数

関数名	説明
NNS_SndArcStrmInit	サウンドアーカイブストリームライブラリを初期化します。
NNS_SndArcStrmStart	ストリームを再生します。

NNS_SndArcStrmStop	ストリームを停止します。
--------------------	--------------

7.5 ストリームライブラリ

ストリーム再生を行うための低レベルライブラリです。関数名は、**NNS_SndStrm** で始まります。

通信で送られてくるデータのリアルタイム再生や、独自のストリームデータフォーマットの波形データをストリーム再生する場合などに使います。

ストリームライブラリの主な関数は、以下の通りです。

表 7-4 ストリームライブラリ関数

関数名	説明
NNS_SndStrmInit	ストリームを初期化します。
NNS_SndStrmAllocChannel	ストリーム再生用のチャンネルを確保します。
NNS_SndStrmFreeChannel	ストリーム再生用のチャンネルを解放します。
NNS_SndStrmSetup	ストリーム再生の準備を行います。
NNS_SndStrmStart	ストリームを再生します。
NNS_SndStrmStop	ストリームを停止します。

7.6 サウンドアーカイブライブラリ

サウンドアーカイブ中のサウンドデータをロードしたり、パラメータなどを取得するライブラリです。関数名は、**NNS_SndArc** で始まります。

サウンドアーカイブライブラリの主な関数は、以下の通りです。

表 7-5 サウンドアーカイブライブラリ関数

関数名	説明
NNS_SndArcInit	サウンドアーカイブを初期化します。
NNS_SndArcLoadGroup	サウンドデータをグループ単位でロードします。

7.7 サウンドヒープライブラリ

サウンドヒープを管理するライブラリです。関数名は、**NNS_SndHeap** で始まります。

サウンドヒープライブラリの主な関数は、以下の通りです。

表 7-6 サウンドヒープライブラリ関数

関数名	説明
NNS_SndHeapCreate	サウンドヒープを作成します。
NNS_SndHeapClear	全メモリを解放します。

NNS_SndHeapSaveState	ヒープの状態を保存します。
NNS_SndHeapLoadState	ヒープの状態を復帰します。

7.8 キャプチャライブラリ

TWL およびニンテンドーDS には、サウンドキャプチャ機能が備わっています。このキャプチャ機能を使って、リバーブなどの効果を出すためのライブラリです。関数名は、NNS_SndCapture で始まります。

キャプチャライブラリの主な関数は、以下の通りです。

表 7-7 キャプチャライブラリ関数

関数名	説明
NNS_SndCaptureStartReverb	リバーブを開始します。
NNS_SndCaptureStopReverb	リバーブを停止します。
NNS_SndCaptureStartEffect	エフェクトを開始します。
NNS_SndCaptureStopEffect	エフェクトを停止します。

7.9 波形再生ライブラリ

シーケンス再生ではなく、波形データを直接再生するためのライブラリです。関数名は、NNS_SndWaveOut で始まります。

マイクでサンプリングしたデータを再生する用途などに使います。リアルタイムに生成される波形データを再生するためには、上記ストリームライブラリを使用します。

波形再生ライブラリの主な関数は、以下の通りです。

表 7-8 波形再生ライブラリ関数

関数名	説明
NNS_SndWaveOutAllocChannel	波形再生用のチャンネルを確保します。
NNS_SndWaveOutStart	波形再生を開始します。
NNS_SndWaveOutStop	波形再生を停止します。

© 2004-2008 Nintendo

任天堂株式会社の許諾を得ることなく、本書に記載されている内容の一部あるいは全部を無断で複製・複写・転写・頒布・貸与することを禁じます。