ニンテンドー 3DS CPU プロファイラマニュアル

ニンテンドー 3DS CPU プロファイラ

2015-04-22

Version 4.04

任天堂株式会社発行

目次

1 概要

ニンテンドー 3DS CPU プロファイラは、多くの CPU リソースを占有する関数を検出するための統計的サンプリングプロファイラです。また各関数を 1 つずつ計装し、CPU リソースの消費を的確にかつ正確に計測する機能も備えています。

1.1 インストール方法

プロファイラパッケージを任意のディレクトリで展開します。

Nintendo 3DS 固有のインストール方法

SDK 改訂に対応する `runtime\CTR\SDK-`*`SDKVersionNumber`* フォルダを開き、その内容を SDK のインストール先にコピーしてください。使用中の SDK に一致する *SDKVersionNumber* がない場合は、使用中のバージョン以前のものから最も近いバージョン番号のフォルダを選んでください。例えば、SDK 7.*x*を使用している場合は、`SDK-4_1_0` フォルダの内容を使用してください。

プロファイラを開発機にインストールする方法の詳細は設定を参照してください。

1.2 必要なシステム

プロファイラの操作には以下の環境が必要です。

必要なシステム

- Windows 7 64-bit
- DirectX 10
  - 各必須ライブラリに別々に DirectX End-User Runtimes (June 2010) をインストールしておいてください。DirectX インストーラは http://www.microsoft.com/download/en/details.aspx?id=8109 からダウンロードできます。
- .NET 4.0 Runtime（Microsoft.comで「.net4.0 runtime」を検索するか、http://www.microsoft.com/ja-JP/download/details.aspx?id=17851 を参照してください。）

Nintendo 3DS 固有の動作条件

- SDK 7.1.1 以降
- ファームウェア 0.20.51 以降
- 以下のいずれか。
  - PARTNER-CTR バージョン 5.70-035 以降および HOSTIO Daemon バージョン 1.03.014 以降
  - IS-CTR-DEBUGGER および IS-CTR-HIO バージョン 2.01 以降
- プロファイラでは、C++ 関数名をデマングルするためにコンパイラを利用します。Windows 内のコンパイラ用環境変数から位置を検索します。本機能を利用するには、ARMCC 5 または 4.1 をインストールしてください。双方がインストールされている場合は、ARMCC 5 のデマングラを利用します。
- プロファイラでは、「アセンブリ」タブで一部の Cygwin の機能を利用します。具体的には、プロファイラが `addr2line` アプリケーションにアクセスするためには、このアプリケーションが入っている Cygwin の `binutils` パッケージをインストールする必要があります。

1.3 注意事項

プロファイラにはいくつかの注意事項があります。

- プロファイラにスレッドスタックを読み込んで解析する必要があります。そのためプロファイラが対応可能なスタックは 64 KByte 以下となります。64 KByte 以上の場合は最も新しい 64 KByte のみを解析するため、UNKNOWN STACK DETAILS（詳細不明なスタック）という結果となる場合や、コールスタックが短縮されてしまう場合があります。
- 実行ファイル中にプロファイラコードが入ったままゲームを出荷することは禁止されています。マスター提出の際にはプロファイラ関係のコードをすべて無効にしてください（ニンテンドー 3DS CPU プロファイラを無効にする方法に関しては、ニンテンドー 3DS CPU プロファイラゲーム API を参照）。また、プロジェクト中にプロファイラライブラリへのリンクを残したままにしないようにしてください。

1.3.1 ホームメニューとの併用

プロファイラではホームメニューは使用できません。アプリケーションがホームメニューから起動した場合は黒画面が表示されます。プロファイル中は必ずテストメニューを使用してください。テストメニューに切り替える場合は、Config ツールを開き、その他の設定からメニューを選択します。ホームメニューからテストメニューへ変更してから、開発機材を再起動してください。開発機材がテキスト表示のテストメニューで起動します。プロファイルが終了したら、同じ手順でホームメニューに変更できます。

1.4 リソース

プロファイラはシステム内の空きリソースの一部を使用します。これにより、プロファイリング中はゲーム内で使用できるリソースの量が減少します。使用するリソース一覧は以下になります。

- *BufferSize* + メインメモリの 600 KByte（*BufferSize* は GUI で設定されているサイズです）。
- `nn::os::Thread` オブジェクト:3 つ + 使用コア数
- `nn::os::Event` オブジェクト:4 つ + 使用コア数

2 プロファイルの取得

2.1 設定

ゲームにプロファイラを初期設定する場合は、以下の手順で行って下さい。

(1) デバッガを開始します。

(2) プロファイラを NAND に保存します。

- **PARTNER-CTR Debugger** の場合：コマンドウィンドウで `nand write` *`PathToProfilerCIA`* と入力します（例えば、`nand write C:\profiler\runtime\CTR\SDK-4_1_0\profiler.cia`）。これにより `profiler.cia` ファイルが NAND に書き込まれます。その後デバッガと開発機材を再起動してください。
- **IS-CTR-DEBUGGER** の場合：Open ダイアログで、ファイルタイプのリストボックスの中から CIA を選択し、`profiler.cia` を参照してください。

(3) オプション: ニンテンドー 3DS におけるフレームマーキング の手順を参考にして、アプリケーション内でフレームをマーキングしてください。

2.2 プロファイル

プロファイルの取得方法は以下の手順となります。問題が発生した場合は、注意事項またはトラブルシューティングを参照してください。それでも問題が解決しない場合は、弊社窓口までお問い合わせください。

(1) デバッガを起動して開発機材に接続します。

(2) `bin` フォルダ内の `Nintendo CPU Profiler.exe` を実行して、プロファイラを起動します。

(3) リボンバー上の青い「接続」ボタンをクリックして、プロファイラを同期させます。

- 表示されるダイアログボックスの指示に従いファイルを指定します。

(4) 「サンプルプロファイル」タブでサンプリングストラテジー、レートおよびその他のオプションを選択します。

(5) 緑の「開始」ボタンをクリックしてプロファイリルを開始します。

(6) プロファイリングを停止します。

- マニュアルで停止するには、赤の「停止」ボタンを押します。
- 指定した時間の経過後に自動的に停止させるには、「開始」と「停止」ボタンの右側にあるドロップダウンリストで時間を選択します。
- または記録バッファが一杯になるとプロファイリングは自動的に停止します（しばらく時間がかかることがあります）。

(7) 「関数」タブ、「情報」タブ、「コールツリー」タブ（「コールスタック」モードが有効な場合のみ）、「サンプルグラフ」タブ、および「コードカバレッジ」タブでプロファイル結果を確認します。

(8) オプション: 「保存」ボタンをクリックしてプロファイルを保存します。

3 ユーザーログデータおよびイベント

プロファイラは手動での統合をほとんど必要とせずに使用できますが、ユーザーログデータやイベントを使用してプロファイルを強化することで、より有効な結果を得ることができます。選択可能なオプションは以下の通りです。

3.1 ハートビート（フレーム）

長時間にわたるデータを分析する場合は、データをフレーム毎などの意味のあるグループに分割すると便利です。複数のシステムや異なるコアはそれぞれ異なるリズムで動作するため、ここではハートビートといったより広範囲の用語を使用して、追跡する際に便利な定期的に発生するリズムおよびフレームを表しています。

プロファイル中にハートビートが記録されると、その記録結果は「ハートビート」タブのドロップダウンボックスの 1 つに表示されます。結果を選択すると、該当するコアに対するハートビートがグラフ表示されます。またこのとき、別の 2 つのハートビートが以下のドロップダウンボックスに表示されます。「固定間隔60Hz」と「固定間隔 1ms」。これらのハートビートはプロファイルデータ内で実際に記録されたものではありませんが、これを選択することでタイミング情報を特定のコアにオーバーレイすることができます。またこの 2 つのハートビートがデータと相関する可能性は低く、タイミングリファレンスとしてのみの提供となります。

特定のハートビートフレームを確認するときは、対象のフレーム上でクリック アンド ドラッグして選択するか、または「ハートビート」タブの特定フレームレート（「60Hz」トグルボタンなど）をクリックして選択します。

3.1.1 ニンテンドー 3DS におけるフレームマーキング

プロファイル取得のためにゲームを修正する必要はありませんが、プロファイラに通知するためのコードを追加することで詳細なフレーム情報を追跡することができます。そうすることでより有効なプロファイル結果の分析と解析が可能になります。

> 補足： 計装プロファイルの取得には、API から関数を少なくとも 1 つ使用する必要があります。

(1) メインループの先頭で `nnprofRecordTopMainLoop` を呼び出してください。

(2) ゲームが VBlank 待ちを開始する前に `nnprofRecordWaitForVBlank` を呼び出してください。

(3) プロファイラヘッダをインクルードします。

```
#include <nn/prof.h>
```

(4) プロファイラライブラリに対してリンクさせます。

```
LIBS += profiler
```

3.1.1.1 フレーム自動検出

ゲームによる明示的なフレーム境界線表示がない場合などに備え、プロファイラにはフレーム自動検出機能があります。ただし、この補助的機能に頼ることは推奨していません。この補助機能では `nngxWaitVSync` への呼び出しを使用してフレーム境界線を設定しますが、手動のフレームマーキング作業と比較すると正確さが劣ります。フレームの先頭としてこの呼び出しを使用するのは正しくありません。プロファイル分析の際にもこの点に十分考慮してください。

プロファイラがゲーム内で nnprofRecordTopMainLoop 関数を検出しない場合は、自動フレーム検出モードが有効になります。

ニンテンドー 3DS におけるフレームマーキングで説明しているとおり、ゲーム内でフレームの先頭および VBlank を手動でマークしておくことを強く推奨します。これによりどのプロファイル内でも両者がマーキングされて表示されるようになります。

3.1.2 推測ハートビート

メインのハートビートをログすることはもちろん重要ですが、プロファイラでは関数の挙動に基づいて各コアのハートビートを推測します。周期性のある関数が存在する場合は、最も周期的な関数を該当コアでのハートビートリズムの目安として使用します。フレームごとにより多くの分析データがあるので、推測ハートビートはより高いサンプリングレート（例：100x かそれ以上）で検知される可能性が高いです。この推測ハートビートは「ハートビート」タブで選択可能です。また「情報」タブには検出された推測ハートビートの一覧が表示されます。ただしこのようなハートビートはデータに基づいた推測であるため、概念的なフレームの先頭と一致することはほとんどありません。この点が問題となる場合は、フレームの先頭を正確にマークするように明示的にハートビートをログすることを推奨します。

3.2 マーク付きコードブロック

ゲームシステムにタイマーを使用してゲームプレイ中の正確な実行時間を簡単に追跡可能にしておくことが、ゲーム開発者の間では一般的です。プロファイラにはマーク付きコードブロックの記録機能があり、実行時間と共に CPU パフォーマンスカウンタを追跡する機能も提供しています。プロファイラを使用すると、プロファイル結果と共にこれらメトリックスのすべてをグラフ表示することが可能です。コードブロック機能の統合に関する詳細は、ニンテンドー 3DS CPU プロファイラゲーム API ドキュメントを参照してください。

コードをコードブロックと一緒に手動で計装した後は、必ずプロファイラのコードブロック記録機能を有効にしてください。有効にするには、「計装プロファイル」タブの「マーク付き」ボタンを使用してください。それからマーク付きコードブロックと共に記録するパフォーマンスカウンタグループを選択します。

プロファイル記録後、マーク付きコードブロックは「計装」タブに一覧表示されます。またこれを「計装」タブ上で選択すると、「サンプルグラフ」内でグラフ表示されます。

4 プロファイラのワークフロー

プロファイラにはトラブルの発見や追跡に利用できる様々なワークフローがありますが、特に便利な使用事例を以下に紹介しておきます。

4.1 ニンテンドー 3DS におけるワークフロー制限

以下に記載するワークフローの説明の一部では、サンプリングレート「1000 回（各フレーム）」の選択が要求されています。別のケースでは、「コールスタック」プロファイルが要求されています。ニンテンドー 3DS では、上記の 2 つのオプションによってプロファイラオーバーヘッドが増加し、ゲームのタイミングが変化することになります。より高いサンプリングレートで、固定時間内で実行される操作の処理時間がさらにかかることになります。標準例として音声システムがありますが、割り込みが 4.889 ミリ秒ごとにハンドリングされなければなりません。

4.2 コードの流れを迅速に理解

「分析」リボンバーの「ストーリー分析」ボタンを使用すると、フレーム内のコードの流れを迅速に理解できます。この方法がフレーム内の動作を理解する際の一番の早道です。

設定

- 「1000 回（各フレーム）」以上で記録します。
- 「コールスタック」のプロファイルを記録します。
- 特定フレームを確認するので、長時間のプロファイルは必要ありません。

手順

(1) 「関数」タブで、コアフィルターを使用して調査対象のコアを選択します（コアのフィルタリングと列見出しで説明）。

(2) リボンバーの「分析」タブで「ストーリー分析」グループの「簡潔」ボタンをクリックします。

(3) 「サンプルグラフ」タブで、平均フレームまたは理解を深めたいフレームをズームインします。この時点でフレームの開始点から終点までの挙動が確認できます。

> ヒント：データをなるべく見やすくするためのヒントをいくつか以下に挙げておきます。
>
> - 「サンプルグラフ」ツールバーの「ロード」ボタンをクリックすると、「フレーム単位負荷」グラフが一時的に非表示になります。ボタンを再度クリックすると表示状態に戻ります。
> - 「サンプルグラフ」ツールバーの「サンプル」ドロップダウンを使用して、サンプルの縦方向の間隔を調整します。縦の間隔が大きいほど、サンプルを左側にある関数名へと目視で追跡することがより簡単になります。
> - サンプルを時間順に並び替えるときは、「サンプルグラフ」を右クリックして、を選択します。
> - 「サンプルグラフ」の関数列をダブルクリックすると、関数の子関数へとすばやく掘り下げることが可能です。展開表示を元に戻すには、同じ関数列を選択したまま右クリックして「直下の子関数の選択解除」を選択します。

(4) オプション: 確認中のフレームを選択して「簡潔」を再度クリックします。これにより対象フレーム内で注意を引く関数のみがそのフレーム用に正確に並び替えられて表示されます。フレームを選択するには、「サンプルグラフ」ツールバーの左端にあるドロップダウンマウスモードから「フレーム選択」に変更して、希望のフレームを 1 回クリックします。

(5) さらに詳細なストーリーを確認する場合は、「ストーリー分析」グループの「フル」、「微細」、または「上位」をクリックします。

また「ストーリー分析」グループ左側にある設定の調整や、名前をクリックしてオプションを拡張することで特定の「ストーリー分析」ボタン内のパラメータを調整することも可能です。この方法の詳細は、ストーリー分析を参照してください。

4.3 スパイク発生率が高い関数を検索

スパイクが発生する関数をワンクリックで検索可能です。

設定

- 「1000 回（各フレーム）」以上で記録します。
- 「コールスタック」のプロファイルを記録します。

手順

「分析」リボンバータブの「スパイク検出」グループ内で「低頻度」ボタンをクリックします。

> ヒント：「低頻度」ラベルをクリックして、オプションを展開したり「最大スパイク数」値の増加します。「低頻度」ボタンを再度クリックしてプロファイルの再分析を行います。

4.4 遅延フレームの原因究明

フレームごとに十分なサンプル数があれば、遅延フレームの正確な原因をボタンのワンクリックで究明できます。

設定

- 「1000 回（各フレーム）」以上で記録します。
- 「コールスタック」のプロファイルを記録します。
- 遅延フレームを確認するために十分な長さのプロファイルを記録します。

手順

(1) オプション:「関数」タブで、調査対象のコアを選択してから「問題フレーム」をクリックします。

(2) 「分析」リボンバータブの「スパイク検出」グループ内で「問題フレーム」ボタンをクリックします。

> ヒント：「問題フレーム」オプションを拡張すると、他のオプションも利用できます。
>
> - 検出する問題フレームの数が増加します。
> - 「選択フレームを使用」を有効にし、対象のフレームを選択してから（マウスモードを参照）「問題フレーム」をクリックします。これでその他のすべてのフレームと比較した場合に、該当フレーム内でパフォーマンス低下がみられる関数のみが選択されます。

4.5 特定時におけるサンプル済み関数をすべて検索

特定時にサンプリングされた関数について調査を行いたい場合があるかもしれません。例えば、CPU パフォーマンスカウンタが特定時においてパフォーマンスの悪い IPC を示している場合などです。

設定

- 「コールスタック」のプロファイルを記録します。
- 特定の時間帯を確認するため、長時間のプロファイルは必要ありません。

手順

(1) オプション:「関数」タブで、すべての関数の選択を解除します。これにより「サンプルグラフ」内の不必要な関数を整理できます。

(2) 「サンプルグラフ」タブで、調査したい時間帯を選択します。

(3) 「サンプルグラフ」を右クリックして、を選択します。

4.6 サンプルグラフでサンプリング実行時間を計測

「サンプルグラフ」でサンプルの実行時間を計測するには、「サンプルグラフ」ツールバーでマウスモードを「測定時間」に変更します(左端)。

設定

- 「サンプルグラフ」ツールバーでマウスモードを「計測時間」に変更します(マウスモードを参照)。

手順

「サンプルグラフ」でサンプル開始点を左クリックして、サンプルの終点へマウスをドラッグします。

> 留意： マウスモードによってマウスの挙動が変わるとともに、タスクを実行するためのショートカットもマウスモードによって異なります。
>
> - 「計測時間」マウスモードでは Ctrl キーを押したまま左クリック アンド ドラッグするとウィンドウをスクロールできます。
> - 「スクロール」マウスモードでは Ctrl キーを押したまま左クリック アンド ドラッグすると時間を計測できます。
> - 選択状態のまま複数の領域を選択するには、「時間選択」マウスモードを使用します。この場合、選択領域に基づいて「関数」および「コールツリー」タブ内の件数の再分析が行われることに注意してください。

計測時間は選択箇所の上部に表示されます(もし「フレーム単位負荷」が非表示のときは、計測時間は選択箇所の下部に表示されます)。左マウスボタンを離すと計測領域は見えなくなります。

5 リボンバー

この章ではリボンバーメニューの中の様々なコントロールや設定について説明します。

リボンを非表示にするには右上にある上矢印をクリックしてください。一度非表示にするとリボンバーはカーソルをリボンバータブの上に移動したときにのみ表示されます。リボンを表示したままにするには右上の下矢印をクリックします。

5.1 ホームタブ

「ホーム」タブには開発機材への接続やプロファイル操作に関するプロファイラの主機能があります。

5.1.1 接続

開発機材に接続するには、まず「接続」ボタンを押してください。このボタンを押すと以下の 3 つの接続処理を開始します。

(1) 開発機材を選択します。利用可能な開発機材がひとつしかない場合は、その開発機材に自動接続します。複数の開発機材が利用可能な場合は、開発機材の選択ドロップダウンメニューのあるウィンドウが表示されます。

(2) プロファイルを取得する実行ファイルを選択するよう促されます。これにより関数アドレスと関数名の相互参照を行います。

(3) 開発機材を ping し、確保したバッファサイズが PC に送信されます。

接続処理が完了すると、プロファイル「開始」ボタンおよび「切断」ボタンが有効になります。

別の実行ファイルのプロファイルを取得したい場合、または別の開発機材に接続したい場合は「切断」ボタンを押してください。

現在の接続情報はアプリケーションの一番下にあるステータスバーに表示されます。

ニンテンドー 3DS CPU プロファイラには「再読み込み」ボタンがあります。「再読み込み」ボタンは接続プロセス完了後にのみ有効になります。実行ファイルをリロードしたい場合に（例えばコードの一部修正、再コンパイル、新しいビルドのプロファイル取得を行う場合）、「再読み込み」ボタンを押してください。

最後に、IS-SNAKE-BOX を使用する時「強制互換」というボタンがあります。これは「強制 CTR 互換モード」を短くしたものです。このボタンは「**IS-SNAKE-DEBUGGER** ソフトで「強制 CTR 互換モード」を選択した場合と同等の操作となります。拡張アプリケーションは、「強制 CTR 互換モード」を使用することにより CTR 実機上で動作させた場合と、SNAKE 実機上で動作させた場合で動作速度が異なるため、拡張アプリケーションの CTR 上での速度を IS-SNAKE-BOX 上で確認できます。これにより、IS-SNAKE-BOX 1 台で CTR と SNAKE それぞれの挙動を確認できます。

5.1.2 バッファサイズ

スピナーで 1 MByte から 16 MByte までのバッファサイズを選択できます。バッファサイズ選択の上にはバッファが一杯になるまでの推定時間が表示されます。これはサンプリングレートとバッファサイズを基に算出した最長プロファイリング時間です。プロファイラ起動中にバッファサイズを変更したときは「再読み込み」ボタンでプロファイラを再起動する必要があります。

一部のケースでは、「接続」実行後にのみ「バッファサイズ」スピナーが利用可能になる場合があります。これは、初めてプロファイラがアプリケーションに接続した際は、デフォルトバッファサイズの 1 MByte が設定されているためです。バッファサイズ値が以前変更されている場合は、変更内容が設定に保存されて、保存されたバッファサイズが使用されます。

> 補足：バッファメモリはゲーム実行前に確保されるので、ゲーム用メモリは減少することになります。必要に応じて Config ツールでメモリモードを変更し、ゲーム用のメモリをさらに大きくパーティションを切り直すことを検討してください。プロファイラのメモリ要件に関する情報は、必要なシステムを参照してください。

5.1.3 ファイル

「開く」ボタンは前回保存したプロファイルを開きます。

「保存」ボタンは現在のプロファイルをディスクに保存します。

「スクリプト」ボタンは自動プロファイリングを行うスクリプトの読み込み、および実行を行います。

5.1.4 ヘルプ

「マニュアル」ボタンを押すと本ドキュメント（ニンテンドー 3DS CPU プロファイラマニュアル）が開きます。プロファイラはプロファイラ実行ファイルの 1 つ上のディレクトリにマニュアルが保存されていると想定して検索します。もしマニュアルが検索できない場合は、エラーメッセージが表示され、マニュアルのファイル名および想定される格納先を表示します。

5.1.5 単位

それぞれのボタンで各関数に要した時間を表す単位をコントロールします（サンプリングしたプロファイルのみ）。選択した単位は、「関数」タブや「コールツリー」タブのようなプロファイルデータを含むタブ、または「サンプルグラフ」タブのツールチップに反映されます。複数の単位を使い分けることにより、関数の様々な分析や比較が可能となります。

パーセント

選択時間枠の関数の実行時間を、その他すべての関数と比較したパーセント値です。例えばサンプル合計数 400 回のうち、ある関数が 100 回サンプリングされていたら、この関数が実行時間の 25% を占有していたことが確認できます。「パーセント」は、関数の処理負荷を相対的に比較する場合に有効です。値の誤差を確認する場合は「エラー」ボタンを有効にします。

時間

各関数の実行時間の近似値（タイマー実測値ではなく、統計的算出に基づいた近似値）です。複数フレームが選択されていたり特定時間が選択されている場合は、時間は各選択内容に対する近似値になります。フレームや時間が未選択の場合の表示値は、プロファイル全体の持続時間に基づいた推定値になります。

実際の関数の実行時間を把握するために「時間」を使用します。一定の時間内またはフレームレート内に収めたい場合に便利です。例えばこの機能を利用して特定のモジュールまたはシステムが一定の時間を越えているかどうかを測定できます（例：人工知能更新ではフレームごとに 2 ミリ秒）。「時間」の値は、サンプリングレートをかなり低速にして、長時間サンプリングを長時間行うとより正確になります。プロファイラのオーバーヘッドが増加するにつれて「時間」の精度が低下します。そのためこの測定基準を使用する場合はサンプリングレートを低く設定することが非常に重要となります。また、個別のフレームを選択すると誤差が増大するため、精度・正確度が低下することにも注意してください。誤差を確認するには「エラー」ボタンを有効にしてください（このエラー算出ではプロファイラオーバーヘッドは考慮に入れておらず、統計的な変分計算になります）。

平均値

各関数の実行時間をフレーム数で割った近似値です。対象コアにフレームが存在しない場合は、「時間」値と等しくなります（平均値は取得されません）。コアにフレームは存在するがフレームが未選択の場合は、実行時間をそのコア上の記録フレーム総数で割った値になります。対象コアに選択フレームが存在する場合は、その

選択フレーム上の時間を現在選択しているフレーム数で割った時間になります。誤差を確認するには「エラー」ボタンを有効にしてください。

> 補足：「平均値」を選択すると「関数」タブの「合計」と「セルフ」列の並べ替え機能が作動していないように見えることがあります。これは、異なったコアの関数を異なったフレーム数で割り出していながら、並べ替えは重要性の順に（絶対時間を基準として）行われるためです。しかし、対象コアのすべての関数に並び替えの順序は正確に表示されています（コア別フィルターをかけると、並べ替えの順序は正しく表示されていることになります）。

サンプル

関数で実際に取得したサンプルの数です。上記の各単位を計算するための基本単位になります。誤差を確認するには「エラー」ボタンを有効にしてください。

エラー

このボタンは他の「単位」ボタンとは異なり、「パーセント」値、「時間」値、「平均値」値、「サンプル」値の横に誤差を表示するかどうかをコントロールします。誤差は結果の精度を判断するために有効です。例えば 1 つのフレームを選択した場合、誤差が非常に大きくなる傾向があります。結果として、「パーセント」値や「時間」値が正確であるように見えても（例えば 8.2%）、誤差で厳密な精度が確認できるため（例えば ±5.8%）、値同士の比較やパフォーマンス評価の際に重要になります。「エラー」ボタンの状態はプロファイラ設定に保存されるので、次回プロファイラを再起動する際に前回選択した状態で動作します。

5.1.6 カラーコーディング

「カラーコーディング」グループはプロファイラ内で選択された関数のカラースキームを操作します。

関数

「関数」は、選択されている関数用にそれぞれ色を割り当てます（色は 15 種類用意されており、15 以上の関数を選択すると同じ色が繰り返し使用されます）。

システム

別のオプションの「システム」を選択すると、選択された各関数には分類設定別に色が割り当てられます。関数は物理、グラフィック、音声といった異なるシステムごとに自動的に分類されます。分類システムの詳細は、システム分類 を参照してください。

選択した関数の色と「アイシクルグラフ」内での色と一致させたい場合は、「カラーコーディング」設定で「システム」を選択してください。

「システム」が選択された場合は、各色分類の説明がコア「サンプルグラフ」内の各コアヘッダの上に表示されます。

5.2 分析タブ

「分析」タブには次の 3 つのツールグループがあり、すばやくプロファイラ分析する場合に便利です。グループには「ストーリー分析」、「スパイク検出」、および「クイックフィルター」があります。

5.2.1 ストーリー分析

「ストーリー分析」ツールは関数の選択や配列を行い、フレームの進行状況に関する要約ストーリー（重要な関数のシーケンス）を表示します。

各ツールを使用するには、「コールスタック」プロファイルを取得した上で「サンプルグラフ」内の 1 つのフレームを拡大します。これで特定のコアだけを確認しやすくなります（「関数」タブツールバーで コアを 1 つ選択します）。

ストーリー分析オプション

以下のオプションを使用して選択関数をより詳細に区別していきます。

親関数を除外：デフォルトで対象関数の親関数も選択されます。このオプションを有効にして結果から親関数を除外するように切り替えます。本オプションにおける「上位ストーリー分析」への影響はありません。

不定期な関数を除外：デフォルトでは、すべての関数がストーリーの一部として選択されます。このオプションを有効にして不定期に実行される関数を除外するように切り替えます。

ハートビート一致：デフォルトでは、すべての関数がストーリーの一部として選択されます。このオプションを有効にしてフレームレートと同様の間隔で実行されていない関数を除外するように切り替えます。

5.2.1.1 簡潔ボタン

フレームの進行状況に関する簡潔なストーリーを確認できる関数を選択します。このボタンを押す前にまず 1 つのコアを選択しておくと（コアのフィルタリングと列見出しを参照）、より効果的です。「サンプルグラフ」の 1 つのフレームを拡大して、ストーリーを確認します。

簡潔ボタンオプション

ボタンオプションの「最大」および「最小」設定で、関数の選択を操作します。ここで設定を変更しても、変更後の設定が永続的に保存されるわけではありません。しかし別の方法として 3 つのプリセットグループが用意されています。「ショート」、「ロング」、と「詳細」があります。作成されるストーリーの詳細レベルを各プリセットグループで設定します。ざっくりとしたハイレベル（最低限の主要関数のみ選択）から最も詳細なレベル（多数の関数を選択して極端かつ重要、詳細な情報を表示）間で設定します。

5.2.1.2 上位ボタン

上記の 3 つのボタン（「簡潔」、「フル」、および「微細」）とは異なり、実装しているアルゴリズムも多少異なります。このボタンは上位関数を選択し、各設定に基づいて特定の関数を除外した後に、選択関数を時間順に配列します。

上位ボタンオプション

「上位」ボタンオプションの設定は関数の選択を操作するときに便利です。「最大結果」設定は関数の選択件数の操作に非常に有効です（しかし関数の除外設定を含む別のオプションに基づき、最終結果は選択件数より少ない件数になることがあります）。もし結果に類似する関数が多数含まれているときは、「類似の子関数を除外」または「類似の親関数を除外」トグルボタンを使用して類似関数を削除できます。

5.2.2 スパイク検出

「スパイク検出」ツールを使用すると、問題の可能性がありさらなる調査を行うべき関数をすばやく見つけ出すことができます。

スパイク検出オプション

以下のオプションを使用して選択関数をより詳細に区別していきます。

最大結果:「低頻度」、「バースト」、「フラッター」、または「問題フレーム」のいずれかをクリックしたときに、何件の関数が選択されるかを決定します。1 つの手段として、この数値を非常に低く設定して(例えば 5 または 10)、スパイク数が 1 番多い関数に最大焦点を当てます。別の手段では、この数値を非常に高く設定して(例えば 100 または 200)何か興味深い結果が表示されるかどうか確認します。

類似の子関数を除外:本設定を有効にすると、親関数に非常に類似している子関数を結果から除外します。親関数の名前は各関数を示す上位システムの名前を取る傾向にあるため、どの上位システムでスパイクが発生しているのか理解するのに便利です。

類似の親関数を除外:本設定を有効にすると、子関数に非常に類似している親関数を結果から除外します。この設定はスパイクの直接の原因になっている下位システム関数に焦点を当てるのに便利です。なぜなら 下位システム関数は CPU 時間が実際の作業時間を示しているリーフ関数である可能性が高いからです。

5.2.2.1 低頻度ボタン

「低頻度」ボタンはスパイク数がほとんどない関数を選択します(これらの関数は非常にまれにサンプリングされます)。

低頻度ボタンオプション

デフォルト設定は、プロファイル期間中にのスパイク数が 10 以下かどうかを基準にしますが、このデフォルト設定は「最大スパイク数」オプションで変更可能です。「最小幅」オプションは、スパイク検出に必要なサンプル実行の規模(継続するサンプル件数)を操作します。「最小幅」を調整してスパイク検出件数の感度設定を増減できます。

5.2.2.2 バーストボタン

「バースト」ボタンは不定期にバーストする関数を選択します(各フレーム呼ばれる傾向がない関数)。

バーストボタンオプション

「不定期」オプションは 関数が検出されるように、フレーム間での不定期率のしきい値を操作します(大きい数値はより不定期であることを示します)。「最小幅」オプションは、スパイク検出に必要なサンプル実行の規模(継続するサンプル件数)を操作します。「最小幅」を調整してスパイク検出件数の感度設定を増減できます。

5.2.2.3 ワイルドボタン

「ワイルド」ボタンは関数の寿命期間を通して変化が激しい関数を選択します。

ワイルドボタンオプション

最初の「最小変化率」オプションは、関数検出に必要となる、フレーム全体で変化する最低合計ミリ秒数を操作します(この数値が減少すると感度がより増加します)。最初の「最小変化率」オプションは、関数検出に必要となる、フレーム全体で変化する最低合計ミリ秒数を操作します(この数値が減少すると感度がより増加します)。

5.2.2.4 フラッターボタン

「フラッター」ボタンは各フレームで定期的に呼び出されているが、フレーム毎に揺れの大きい関数を選択します。

フラッターボタンオプション

「不定期」オプションは 関数が検出されるように、フレーム間での不定期率のしきい値を操作します(大きい数値はより不定期であることを示します)。最初の「最小変化率」オプションは、関数検出に必要となる、フレーム間で変化する最低ミリ秒数を操作します(この数値が減少すると感度がより増加します)。2 番目の「最小変化率」オプションは、検知されるために必要となる、フレーム間で変化する最低パーセント値を操作します(この数値が減少すると感度がより増加します)。

5.2.2.5 増減ボタン

「増減」ボタンは各フレームで定期的に呼ばれているが、緩やな増減があり、最終的にプロファイル全体をみると大きな差異が確認できる関数を選択します。例えば、ある関数で最初のフレームでは占有率が 5% で、その後各フレームで徐々に増加して最後のフレームでは 20% になったとします。パーセントには急激な変化はなく、数字はフレームから次のフレームへと徐々に変化していきます。

増減ボタンオプション

「不定期」オプションは 関数が検出されるように、フレーム間での不定期率のしきい値を操作します（大きい数値はより不定期であることを示します）。最初の「最小変化率」オプションは、関数検出に必要となる、フレーム全体で変化する最低合計ミリ秒数を操作します（この数値が減少すると感度がより増加します）。最初の「最小変化率」オプションは、関数検出に必要となる、フレーム全体で変化する最低合計ミリ秒数を操作します（この数値が減少すると感度がより増加します）。

5.2.2.6 問題フレームボタン

「問題フレーム」ボタンは、1 番遅いフレーム（複数）を選択した後に、他のすべてのフレームと比較した際にパフォーマンスがさらに悪いフレームを選択します。

問題フレームボタンオプション

「最大フレーム」オプションは、分析用遅延フレームの選択件数を操作します。「選択フレームを使用」オプションは、分析用にユーザーが選択したフレームを使用するので、問題があると思われる特定のフレームやフレームのセットに焦点を当てることができます（選択フレームがない場合は、本オプションは無効であるとみなして「問題フレーム」分析を行います）。

5.2.3 クイックフィルター

「クイックフィルター」セクションでは、複数の異なる測定基準でプロファイルの分析をすばやく行うことが可能です。

「クイックフィルター」の機能は手動で行うことも可能ですが、「クイックフィルター」ではワンクリックですばやいプロファイル分析が可能になります。クイックフィルターボタンが押されると、まず以下の処理を実行して GUI のステートをリセットします。

- すべての関数の選択を解除。
- 「関数」タブ：「コアすべて」および「すべてのスレッド」を選択、フィルターをクリア、関数を「セルフ」ヘッダによって分類。
- 「サンプルグラフ」タブ：「コールスタック」プロファイルの場合は「合計」を選択、それ以外は「セルフ」を選択。

現在、すべての「クイックフィルター」では以下のステップを実行します。

(1) 「関数」タブで、「コールスタック」プロファイルの場合は「合計」ヘッダが選択され、それ以外では「セルフ」ヘッダが選択されます。

(2) 「関数」タブで、「RegEx」が有効な状態でフィルターテキストが特定の文字列に設定されます。これで押された特定ボタンの設定に関連するすべての関数を分類します。将来的には各クイックフィルター文字列をカスタマイズできるようになる予定ですが、本バージョンでは「クイックフィルター」のいずれか 1 つをクリックした後に「関数」タブのテキストフィルターを自由に変更することができます。

(3) 「サンプルグラフ」タブ内で、「複合のみ」が選択されます。選択されたすべての関数をコアごとに 1 つのロードラインとしてまとめて表示します。

(4) 「関数」タブで、「上位選択」ボタンが上位 45 件の関数を選択するように設定されます。

アイドル時間

「アイドル時間」フィルターは各コアのフレーム毎のアイドル時間を表示します。ある程度のアイドル時間はすべてのゲームにおいて発生しますが、過剰なアイドル時間はコアの利用状況が非効率であることを示します。

物理

「物理」フィルターは各コアの 1 フレームあたりの物理的時間を表示します（Havok および Bullet ミドルウェアを含む。)

GFX

「GFX」フィルターは各コアの 1 フレームあたりのグラフィックスの時間を表示します。このフィルターには「パーティクル」フィルターも含まれます。

パーティクル

「パーティクル」フィルターは各コアの 1 フレームあたりのパーティクルシステムの時間を表示します。

アニメーション

「アニメーション」フィルターは各コアの 1 フレームあたりのアニメーションの時間を表示します。

人工知能

「人工知能」フィルターは各コアの 1 フレームあたりの人工知能の時間を表示します（挙動およびナビゲーション）。

音声

「音声」フィルターは各コアの 1 フレームあたりの音声時間を表示します（Audiokinetic、Ogg Vorbis、および FMOD ミドルウェアを含む）。

オンライン

「オンライン」フィルターは各コアの 1 フレームあたりのオンライン時間を表示します（Quazal ミドルウェアを含む）。

ジョブシステム

「ジョブシステム」フィルターは各コアの 1 フレームあたりのジョブシステム時間を表示します。

演算

「演算」フィルターは各コアの 1 フレームあたりの演算時間を表示します。

メモリ

「メモリ」フィルターは各コアの 1 フレームあたりのメモリ時間を表示します。

リセット

「リセット」クイックフィルターボタンは本セクションの冒頭で説明したとおり、リセットを実行します。各種「クイックフィルター」を使用した後に GUI をデフォルトの状態に戻すときに使用します。最も重要な点として「サンプルグラフ」で「複合のみ」設定が「複合グラフ非表示」設定に戻ります。

5.2.3.1 クイックフィルターのカスタマイズ

すべてのクイックフィルターはカスタマイズが可能で、新規作成も可能です。「ディレクトリ」タブでクイックフィルターの読み込み先フォルダに「ボタン:」を設定することができます（ボタンディレクトリを参照）。

既存のクイックフィルターの動作変更

クイックフィルターのデフォルト動作を変更する場合は、オリジナルの Quick Filter XML ファイルをクイックフィルターフォルダ（`bin\Quick Filters`）からカスタムボタンディレクトリへコピーします。これでクイックフィルターの動作を各アプリケーションに合わせて編集できます。ボタン読み込みの際に従来のデフォルト版ボタンの代わりにカスタムボタンを読み込みます。

クイックフィルターの新規作成

新規のクイックフィルターを作成するには、まず既存の Quick Filter XML ファイルのうち 1 つを自分のカスタムボタンディレクトリにコピーしておくことを推奨します。それからボタンのフィールドを編集して希望の動作を設定します。

ここで特に重要な点は XML ファイル内の <Title></Title> フィールドを変更することです。Title フィールドは各ボタンの性質を決定するために使用されます。もし Title が別の Title と一致した場合は、新規ボタンが既存ボタンの内容を上書きします。

システム分類機能の使用

クイックフィルタースクリプトには `Classification.xml` ファイルにある正規表現を使用する機能があります。この機能を使用するとクイックフィルターで分類機能と同じ検索機能を使用できるようになります（分類機能に関する詳細はシステム分類を参照してください）。スクリプト内の分類機能にアクセスするには、標準 Windows PowerShell 変数マークアップで、表現を定義する XML タグと同じ変数名を使用します。

クイックフィルタースクリプトが読み込まれると、シンプル検索を行い、読み込まれた classification 値を対照してスクリプトを置き換えます。したがって、どのようなスクリプトにおいても分類定義に使用されるスクリプトとは異なるフォーマットの変数名を使用することを強く推奨します。デフォルトの分類設定は例えば *$AudioClassification* というように *$<System>Classification* のフォーマットを使用しています。

5.3 ハートビートタブ

「ハートビート」タブでは、プロファイルデータをコアごとの特定ハートビートに対してフレーム分けすることができます。また、特定レートのフレームのサブセットもすばやく選択できます。

5.3.1 グループ選択

グループ選択ではそれぞれのレートにあわせた特定のフレームグループの選択が可能です。以下の表で各ボタン操作を説明します。

表 5-1：グループ選択ボタンとその操作

ボタン	操作説明
60Hz	60Hz 以上（16.6ms 以下）の速さで動作中のコアすべてのフレームすべてを選択
30Hz	30Hz から 60Hz の間（16.6ms から 33.3ms の間）で動作中のすべてのコア上のフレームすべてを選択
20Hz	20Hz から 30Hz の間（33ms から 50ms の間）で動作中のすべてのコア上のフレームすべてを選択
15Hz	15Hz から 20Hz の間（50ms から 66.6ms の間）で動作中のすべてのコア上のフレームすべてを選択
12Hz	12Hz から 15Hz の間（66.6ms から 83.3ms の間）で動作中のすべてのコア上のフレームすべてを選択

ボタン	操作説明
12Hz 未満	12Hz 未満（83.3ms を超える）で動作中のすべてのコア上のフレームすべてを選択
30Hz 未満	30Hz 未満（33.3ms を超える）で動作中のすべてのコア上のフレームすべてを選択
反転	フレーム選択を反転（選択フレームの選択を解除、未選択フレームを選択に変更）
すべて選択解除	すべてのフレームの選択を解除

フレームの選択状態は「サンプルグラフ」タブで確認できます。「反転」および「すべて選択解除」以外のボタンでオンオフ切り替えが可能です。特定のボタンをオンに切り替えると、レートと一致する該当フレームが選択されます（オレンジ色でハイライトされます）。ボタンをオフに切り替えると、そのレートと一致するフレームの選択が解除されます。例えば切り替え操作でフレームレート範囲を選択するといったように、各範囲をそれぞれ選択、選択解除することができるようになります。「サンプルグラフ」内のそれぞれのフレームを直接選択したり選択解除することで、グループ選択をマニュアルでオーバーライドできます。

フレームを選択したり選択解除すると、すべての表示中の値（例えば「関数」タブのパーセント値）は 選択されたフレームを反映するように再計算されます。フレームが未選択の場合は、データセット全体が値に反映されます。

5.3.2 ハートビートコア

プロファイル中に記録されたコアに対して、番号付き「コア」グループが「ハートビート」タブに表示されます。「コア」グループでは、コア別にハートビートのフレームを選択します。また選択フレームの適用をすべてのコアにするか、またはマスターフレームレートとして扱うかどうか切り替えることができます。

ドロップダウンボックスでは 3 つの選択肢が表示されます。

(1) 固定間隔（固定間隔 60Hz、固定間隔 1ms）。
(2) ゲームが直接記録するハートビート（例えば「メインハートビート」）。
(3) 推測ハートビート（ゲーム内で自動検出される周期性リズム）。

固定間隔オプションでは各コアに対する「サンプルグラフ」内で固定時間マーカーを追加してグラフ表示します。仕様では、固定間隔オプションをフレーム境界やフレーム選択に使用することができません。なぜならゲームプロファイルデータは固定間隔設定と同期することがないため、結果が誤解を与える情報になってしまうからです。

記録ハートビートや推測ハートビートでは「サンプルグラフ」上にフレームマーキングを表示するので、この情報をさらなる分析や選択において活用できます。例えば、すべての「30Hz」フレームを「グループ選択」リボンバーグループから選択します。記録したハートビートはニンテンドー 3DS CPU プロファイラゲーム API で説明する関数を使用して、ゲームコード内で手動でログに記録します。推測ハートビートは周期的な関数の動作に基づいて自動検出されます。しかし、サンプリングレートが低すぎたり検出パターンが確認できないときは、推測ハートビートは確認されなかったという結果になります。推測ハートビートには直近に検出された周期的な関数に基づいた任意のフレーム境界線があるため、実際のフレーム開始点と一致することはほとんどありません（正確なフレーム境界線を確認したい場合は意図的にコード内でハートビートを記録することを推奨します）。最後に、推測ハートビートはプロファイルで検出数が高いものから低いものへと順番で表示されるため、ドロップダウンボックス内で上部に表示される推測ハートビートを注意して確認するようにしてください。

記録したハートビートや推測ハートビートが選択されると、「すべてのコアに適用」トグルボタンを切り替えることで対象ハートビートをすべてのコアに対して使用することができます。また、現在選択されているハートビートはマスターフレームとして使用可能で（「トグルボタンで切り替え」トグルボタンで切り替え）、その設定した場合は「サンプルグラフ」の最下部のタイムライン上に描かれます。デフォルトでは、記録されたハートビート（フレームマーカー）はマスターフレームレートとして使用し「サンプルグラフ」タイムラインに表示します。

5.4 サンプルプロファイルタブ

「サンプルプロファイル」の結果に影響がでる設定がいくつかあります。以下の設定を考慮してください。

タブの設定はすべてプロファイル取得関連のため、このタブは開発機材に接続中にのみ表示されます。接続が完了すると、このタブはリボンバーに自動的に表示されて選択可能タブになります。

5.4.1 プロファイルボタン

「開始」および「停止」プロファイルボタンは左端にあります。「開始」ボタンを押すと「停止」ボタンに表示が変わるので、マウスカーソルを移動させなくても手動でプロファイルを停止することができます。

「開始」ボタンの右側の数値はプロファイル経過時間を示します。「開始」ボタンを押すと計測を開始します。

「開始」ボタンの横にあるドロップダウンボックスで、そのプロファイルの停止方法を操作できます。「停止」ボタンでプロファイルを手動停止する場合は、このドロップダウンで「手動ストップ」を選択しておきます。指定の時間だけ記録する場合は、ドロップダウンボックスでその時間を選択します。指定した時間前でもプロファイルの「停止」は可能です。

プロファイルはプロファイルバッファが一杯になった時点で必ず停止します。

> 重要：「サンプルプロファイル」タブの「開始」ボタンを押すと、サンプルプロファイルを開始します（「計装プロファイル」タブのボタンとは異なります）。計装プロファイルを開始する場合は、まず「計装プロファイル」タブに切り替えてから「開始」ボタンを使用します。

5.4.2 サンプリングストラテジーおよびレートボタン

サンプル取得のトリガーイベント、およびイベントトリガー時のレートを判定します。一般的には、以下の 2 種類のイベントがあります。「時間によるサンプル」と「パフォーマンスカウンタによるサンプル」です。

「時間によるサンプル」ではタイマーを使用してサンプル取得のタイミングを判断します（ジッターを使用してサンプリングの質を向上させます）。「パフォーマンスカウンタによるサンプル」では「サンプリングレート」ドロップダウンボックスで指定したカウンタ *n* 個ごとのイベントをサンプリングします。

サンプリングストラテジー ドロップダウンボックス

サンプル取得のトリガーイベントを判定します。サンプリングストラテジーには次のオプションがあります（*n* は「サンプリングレート」ドロップダウンボックスで定義されます）。

時間によるサンプリング

デフォルトのサンプリングオプションです。60Hz の 1 フレームあたり *n* 回サンプルを取得します。

ICache ミスによるサンプル

命令キャッシュミスが *n* 回発生するごとにサンプルを取得します。

DCache 読み込みミスによるサンプル

データキャッシュ読み込みミスが *n* 回発生するごとにサンプルを取得します。

DCache 書き込みミスによるサンプル

データキャッシュ書き込みミスが *n* 回発生するごとにサンプルを取得します。

分岐予測ミスによるサンプル

分岐予測ミスが *n* 回発生するごとにサンプルを取得します。

命令ストールサイクルによるサンプル

メモリから読み込まれる命令を待機するためにストールしたサイクル *n* 回ごとにサンプルを取得します。

D-Hazard ストールサイクルによるサンプル

データハザード(メモリから読み込まれるデータの待機中あるいはデータ演算の待機中)のためにストールしたサイクル *n* 回ごとにサンプルを取得します。

LSU フルストールサイクルによるサンプル

ロードストアユニットが一杯でストールしたサイクル *n* 回ごとにサンプルを取得します。

サンプリングレートドロップダウンボックス

プロファイラでは選択した「サンプリングストラテジー」に基づいた複数のサンプリングレートを選択できます。「時間によるサンプル」を選択した場合、レートは 1 つの 60Hz フレームの間にサンプリングする回数を表します。例えばサンプリングレートが「500 回(各フレーム)」の場合、60Hz の 1 フレームあたり 500 サンプルの取得することになります。ゲームが毎秒 30 フレームの場合、500 回のレートであれば実際にはその 2 倍のレート(各フレーム 1000 回)でサンプリングします。毎秒 20 フレームのゲームの場合、その 3 倍のレート(各フレーム 1500 回)でサンプリングします。サンプリングに「パフォーマンスカウンタによるサンプル」の 1 つを選択した場合、レートはそのパフォーマンスカウンタが基準となります。

時間によるサンプリングの場合は、サンプリングレートが変化するため、プロファイリング予測時間が表示されます。パフォーマンスカウンタによるサンプリングではタイマーを使用しないのでプロファイリング予測時間は表示されません。

5.4.3 サンプルごとのデータ

各サンプルで記録するデータを設定します。

サンプルごとのバイト数は「サンプルごとのデータ」グループの左上に表示されます。この数字は「サンプルごとのデータ」で選択した設定によって変化します。

5.4.3.1 パフォーマンスカウンタドロップダウン

「パフォーマンスカウンタ」ドロップダウンボックスで、各サンプルでどのパフォーマンスカウンタを記録するかを選択します。

利用可能なパフォーマンスカウンタの種類や記録するデータについての詳細はパフォーマンスカウンタグループを参照してください。

5.4.3.2 コールスタック録音オプション

各サンプルでどのような関数データを記録するかには以下の 3 つの選択肢があります。

リーフのみ:サンプルイベントが検出されると、現在実行中の関数のみが記録されます。このオプションで記録されるデータは非常に少ないため、通常プロファイルにもっと時間がかかります。リーフ関数のみを記録する場合、次のような結果が予測されます。

- 利点:最小限のオーバーヘッドが発生します。
- 利点:最小限のデータが記録されるのでより長い時間記録されます。
- 欠点:アプリケーション内で何か起こってしたのかに関する詳細が表示されません。

推測:このオプションは無効に初期設定されています。それを有効にするには、「コールスタック」プロファイルをまず取得する必要があります。サンプルイベントを検出した際、現在実行中の関数のみを最小限の情報とともに記録するので、プロファイル時間を長くとることが可能です。このデータを PC で表示する際、以前の「コールスタック」プロファイルを利用して、このサンプルイベントのコールスタックを推測します。「リーフのみ」と「コールスタック」プロファイリング双方の利点を得ることが「推測」の目的です。

- 利点:「リーフのみ」と「コールスタック」記録オプションの双方の良い面を利用することができます。
- 欠点:コールスタックは推測なので、一部の推測では事実上誤ったコールスタックが報告されることにつながる可能性があります。結果を確認する際、この点に留意する必要があります。

コールスタック:サンプルイベントの検出で、コールスタック全体が記録されます。このオプションでは、記録されるコールスタックの深さによって、不定量のメモリを記録します。コールスタックを記録する場合、次のような結果が予測されます。

- 利点:関数の実行時間の「合計」(関数内の時間およびその関数が呼び出した関数すべての時間)は「セルフ」時間とともに「関数」タブに表示されます。
- 利点:コールツリーは各プロファイルから構成され、「コールツリー」タブに表示されます。コールツリーは各関数の呼び出し元(複数)、またツリーの各関数の実行時間を表示します。
- 欠点:バッファはより速いスピードで一杯になるため、プロファイル可能時間が減少します。
- 欠点:プロファイラからのオーバーヘッドが大幅に増大するため、フレームレートが悪化したりプロファイラの精度にも影響します(増大するプロファイラ処理がキャッシュが悪影響を与えるためです)。

> 補足 : 「推測」プロファイルを取得するためには「コールスタック」プロファイルを 1 つしか必要としませんが、まずは多数の「コールスタック」プロファイルを取得することを推奨します。より多くの「コールスタック」プロファイルがあると、コールスタックを推測するために使用するシステム改善につながります。「推測」プロファイルを取得する前に少なくとも 5 つの「コールスタック」プロファイルを取得するようにしてください。これを自動的に実現するには、スクリプト記述 システムを使用すると簡単です。

5.4.4 コア

本セクションの各種ボタンは、コア別にプロファイルをオン／オフ切り替えできます。プロファイルの取得を 1 つのコアに絞ることでプロファイル時間を長めに設定したり、他のコアに要するプロファイラのオーバーヘッドを縮小することができます。コアは必ず 1 つは選択してください。

5.4.5 スレッドの選択

プロファイルするスレッドを選択します。デフォルトでは「有効」スレッドをプロファイルしますが、これはサンプリングごとに現在スレッド関数またはコールスタックが記録されることになります。ある特定のスレッドだけをプロファイルしたい場合には、ゲームを起動したあとに「スレッド更新」ボタンをクリックしてください。プロファイラはゲームからアクティブスレッドのリストを取得します。ゲームがスレッド一覧で応答するまで少し待ち、スレッド選択ドロップダウンボックスをクリックしてプロファイルするスレッドを選択します。

「有効」スレッドでプロファイルする場合は、プロファイラが最後にアクティブであったスレッドや関数を判定できない状態となる「システムアイドルスレッド」に一部の時間が振り分けられます。プロファイルをシングルスレッドに制限することにより、最後にアクティブであった関数か確認でき、「システムアイドルスレッド」への時間振り分けも行われません。

5.5 計装プロファイルタブ

このタブを使用して「計装プロファイル」を取得します。計装プロファイルを取得する前に、3 つのオプションを選択します。

(1) プロファイルする関数を選択します(「マーク付き」を選択している場合は、関数の指定はオプションです)。

(2) 記録するパフォーマンスカウンタグループを選択します。

(3) 「マーク付き」を有効にしたコードブロックを記録するかどうかを選択します(関数を選択済みの場合はこの指定はオプションです)。

タブの設定はすべてプロファイル取得関連のため、このタブは開発機材に接続中にのみ表示されます。接続が完了すると、このタブは「リボンバー」に自動的に表示されて選択可能タブになります。

5.5.1 計装の準備

計装プロファイル取得を可能にするためには特定の条件を満たす必要があります。

ニンテンドー 3DS CPU プロファイラ で計装するためには、プロファイラライブラリがアプリケーションに関連づけられていること、および少なくとも 1 つの プロファイラ API が呼び出されている必要あります。API には、より正確なフレームレート詳細を取得する仕組みが実装されている `nprofRecordTopMainLoop` を使用することを推奨します。詳細は ニンテンドー 3DS におけるフレームマーキング および ニンテンドー 3DS CPU プロファイラ ゲーム API を参照してください。

5.5.2 プロファイルボタン

「開始」および「停止」プロファイルボタンは左端にあります。「開始」ボタンを押すと「停止」ボタンに表示が変わるので、マウスカーソルを移動させなくても手動でプロファイルを停止することができます。

「開始」ボタンの右側の数値はプロファイル経過時間を示します。「開始」ボタンを押すと計測を開始します。

「開始」ボタンの横にあるドロップダウンボックスで、そのプロファイルの停止方法を操作できます。「停止」ボタンでプロファイルを手動停止する場合は、このドロップダウンで「手動ストップ」を選択しておきます。指定の時間だけ記録する場合は、ドロップダウンボックスでその時間を選択します。指定した時間前でもプロファイルの「停止」は可能です。

プロファイルはプロファイルバッファが一杯になった時点で必ず停止します。

> 重要：「計装プロファイル」タブの「開始」ボタンを押すと、計装プロファイルを開始します(「サンプルプロファイル」タブのボタンとは異なります)。サンプルプロファイルを開始するには、まず「サンプルプロファイル」タブに切り替えてから、そこの「開始」ボタンを使用します。

5.5.3 計装する関数の選択

以下の手順で関数を選択します。

(1) まず「サンプルプロファイル」を取得して、「関数」タブを追加表示させます。

(2) 「追加」ボタンをクリックしてハイライトさせます。

(3) 「関数」タブ、「コールツリー」タブ、または「コードカバレッジ」タブの中からプロファイルしたい関数をクリックします。
その関数名が「計装プロファイル」タブに表示されます。「コードカバレッジ」タブを使用する場合は、画面下のフィルタリングボックスを使用して特定の関数を検索すると便利です。

別の関数を選択する場合は、再度「追加」ボタンをクリックしてから別の関数をクリックします。この場合、「関数」タブ、または「コードカバレッジ」タブのどちらからでも選択できます。関数の選択を解除する場合は、「計装プロファイル」タブの関数名の横にある赤い「X」ボタンをクリックします。

5.5.3.1 計装対象関数の選択における制限事項

計装プロファイルが利用できない関数が一部あります。これはプロファイラにおける計装の仕組み上の制限によるものです。コードの計装に際しては、プロファイラがコードの一部を上書きします。この上書きが確実に動作するように以下の制限事項を設けてあります。

- 関数は ARM コードである必要があります。関数が追加された時点でプロファイラはこの制限を有効にします。
- スタック経由で渡された引数は使用できません。これは計装を正しく行うために命令が上書きされるためです。プロファイラにおけるこの制限は完全に実装されていないため注意が必要です。
- 分岐コードが最初の命令にならないようにします。分岐の多くは、コード中の定位置を基本にしているため、分岐コードを計装した場合正常に動作しません。関数が追加された時点でプロファイラはこの制限を有効にします。
- 最初の命令は計装のために上書きするため PC 相対にはできません。関数が追加された時点でプロファイラはこの制限を有効にします。
- コードは最初の命令へ分岐することはできません。最初の命令は計装のために上書きされるため、コードがこの位置に分岐する場合は正常に動作しない場合があります。この制限は現時点では未実装のため注意が必要です。

> 補足：この関数チェックはゲームやプロファイラでのエラー表示を回避するために安全策を取り失敗する傾向があります。そのため、ほとんど発生することはありませんが、計装可能な関数を誤って無効とマークしてしまう可能性もあります。

5.5.4 パフォーマンスカウンタグループの選択

パフォーマンスカウンタは計装関数やコードブロックとともに記録できます。記録したいカウンタグループを、「パフォーマンスカウンタ」ドロップダウンボックスから選択します。

パフォーマンスカウンタデータの記録処理ではバッファスペースの消費が若干速く、プロファイラのオーバーヘッドも多少増加します。

利用可能なパフォーマンスカウンタの種類や記録するデータについての詳細はパフォーマンスカウンタグループを参照してください。

5.5.5 マーク付きコードブロックの記録

「マーク付き」ボタンを選択してコードブロックの記録をオンにします。コードブロックは必ずゲームのソースコード内でニンテンドー 3DS CPU プロファイラゲーム API を使用して手動で計装してください。このボタンはプロファイル取得中の記録作業をオンにするだけです。

> 注意：コードブロックの記録はプロファイル記録結果に影響することがあります。

5.6 ディレクトリタブ

「ディレクトリ」タブでは、プロファイラがファイル検索に使用する重要ディレクトリをそれぞれ指定することができます。

各ディレクトリは以下の方法で編集可能です。

(1) テキストボックスを直接編集：ディレクトリをテキストボックスにタイプします。ディレクトリが存在しない場合は、テキストボックスの背景色がオレンジに変わります。有効なディレクトリが入力されると、背景が通常に戻ります。無効なディレクトリがテキストボックスに入力されて、テキストボックスから移動すると、ディレクトリは最後の有効ディレクトリに置き換えられます。

(2) ディレクトリを閲覧：テキストボックス右のディレクトリの閲覧ボタンをクリックして、希望のディレクトリを検索します。

補足： ディレクトリをデフォルトに戻す場合は、テキストボックスを直接編集して内容をクリアします。

5.6.1 ボタンディレクトリ

「ボタン」ディレクトリを使用して各カスタム「クイックフィルター」ボタンの読み込み先フォルダを指定します。この指定により、ユーザーがプロファイラバージョン間で簡単に保存可能なカスタムボタンの作成が可能になります。カスタムボタンの作成方法についての詳細は、クイックフィルターのカスタマイズを参照してください。

「ボタンディレクトリ」はデフォルトで未入力です。つまりどのカスタムボタンも読み込みません。

5.6.2 コードディレクトリ

「コード」ディレクトリを使用して「アセンブリ」タブが自身のソースコードの検索に使用するフォルダを指定します。他のマシンで取得されたプロファイルの検証の場合はコードフォルダの位置が一致しないため、この指定方法が便利です。

ソースファイルを検索するためには、プロファイラは Cygwin パッケージ `binutils` 内の `addr2line` ツールを使用して特定アドレスをファイル名や列番号に変換します。「コード」ディレクトリを指定すると「アセンブリ」タブはコードファイルの検索を行いますが、まず「コード」ディレクトリで指定されたフォルダ内のベースファイル名から検索を開始します。検索結果ファイルが見つからない場合は、ファイルが見つかるまで、または追加するディレクトリがなくなるまで継続してベースファイル名からディレクトリを追加していきます。

「コード」ディレクトリが未指定の場合、または「コード」ディレクトリの検索結果がまったく無い場合、「アセンブリ」タブはデフォルトで `addr2line` ツールによって戻されたかのようにファイルの読み込みを試行します。

「コード」ディレクトリはデフォルトで空箱になります。

5.7 クイックリボンバー

クリックリボンバーはプロファイラの右上端にあります。リボンバーが最小化されている場合でも、以下の操作が簡単に実行可能です。

- 開発機材への接続。
- 開発機材からの切断。
- 「サンプルプロファイル」の開始。
- プロファイル取得の停止。
- 現在のプロファイルをディスクへ保存。

6 関数タブ

「関数」タブでは、1 つのプロファイル内でサンプリングしたすべての関数を表示します。

6.1 関数のソート

関数のソートを行うには、「合計」、「セルフ」、「コア」、「スレッド」、「モジュール」、「オペコード」、「名前」のヘッダをそれぞれクリックします。ただし「合計」列は、「コールスタック」プロファイルの取得時にのみ表示されます(コールスタック録音オプションを参照)。「スレッド」、「モジュール」、「オペコード」のヘッダはそれぞれ「スレッド」、「モジュール」、「オペコード」ボタンをクリックしたときにのみ切り替え表示されます。

デフォルトでは、各関数に要した時間を示す「セルフ」で関数をソートします(サブ関数は含まれません)。「合計」ヘッダをクリックすると、各関数の実行時間およびその関数が呼び出したすべての関数の実行時間の合計を示す「合計」でソートします。

6.2 関数タブのツールバー

「関数」タブのツールバーでの操作を以下に示します。

6.2.1 コアのフィルタリングと列見出し

プロファイルに複数のコアが含まれる場合は、ツールバーの「コア」ドロップダウンをクリックして、表示されたリストの中から「コアすべて」やコア別の選択ができます。ただし「コア」ドロップダウンに表示されるのはプロファイル済みのコアのみになります。デフォルトでは「コアすべて」が選択されており、新規のプロファイルを取得したり読み込んだときは、「コア」ドロップダウンメニューがリセットされてデフォルトの「コアすべて」に戻ります。

例えば「コア 1」などの特定のコアを選択すると、表示される関数はすべて同じコアになるため、「コア」列は非表示となります。このオプションは特定のコアに焦点を当てる場合や、タブの横スペースの有効活用に利用できます。

6.2.2 Diff 合計と Diff セルフ列の見出し

デフォルトでは、「Diff 合計」列と「Diff セルフ」列は非表示ですが(横スペース節約のため)、「関数」ツールバーの「Diff」ボタンをクリックして表示することができます。ボタンを再度クリックすると列が非表示になります。

「Diff 合計」列と「Diff セルフ」列は現在のプロファイルを最後に読み込んだプロファイルと比較し、各関数の違いを割合で表示します。例えば、特定の関数の「セルフ」の割合が、最後に読み込んだプロファイルが 10% で現在のプロファイルが 12% の場合、「Diff セルフ」列には、+2% の値が表示されます。また、特定の関数の「セルフ」の割合が、最後に読み込んだプロファイルが 10% で現在のプロファイルが 8% の場合、「Diff セルフ」列には、-2% の値が表示されます。

ゲームのパフォーマンスが経時的にどのように変化したかを見るには、まず以前に取得したプロファイル(例:昨日、一週間前、一ヶ月前)を読み込み、その後現在のプロファイルを取得します。「Diff 合計」列と「Diff セルフ」列内では、各関数が割合としてどれだけ増加または減少したかを表示します。しかし特定の関数の処理時間が短くなるよう最適化した場合(その割合が減少)、割合の点では、たとえ両プロファイルの絶対取得時間が同じであっても、他の関数の処理時間が増加します。割合の絶対値の違いによって全関数をソートするには、「Diff 合計」または「Diff セルフ」列の見出しをクリックしてください。ソートは、正負にかかわらず最も大きな割合の変更があった関数を優先するために、絶対値に基づいて行われます。

補足：「Diff 合計」列はコールスタックプロファイル上でのみ利用可能です。

6.2.3 スレッドフィルタリングおよびスレッド列見出し

デフォルトでは「スレッド」列は非表示となっています(横スペース節約のため)。しかし関数が属するスレッドを確認したい場合は、ツールバーの「スレッド」ボタンを選択します。ボタンを再度クリックすると列が非表示になります。

逆に特定のスレッドの中から関数をリスト表示するには、「スレッド」ドロップダウンボタンをクリックして目的のスレッドを選択します。1 つのスレッドを選択すると、表示される関数はすべて同じスレッド内のものとなるため、「スレッド」列は非表示になります。デフォルトは「すべてのスレッド」が選択されています。新規のプロファイルを取得したり読み込んだときは、「スレッド」ドロップダウンメニューがリセットされて「すべてのスレッド」に戻ります。

デフォルトでは、スレッド名は 32-bit 識別子になります（例えば 0x1052D298）。プロファイラがスレッド名を確認できる場合は、ドロップダウンに表示されます。しかし、「スレッド」列では必ず 32-bit 識別子だけが表示されます。

6.2.4 モジュールの列見出し

デフォルトでは、「モジュール」列は非表示ですが（横スペース節約のため）、「関数」ツールバーの「モジュール」ボタンをクリックすると表示できます。ボタンを再度クリックすると列が非表示になります。

「モジュール」列には各関数のモジュール名が表示されます。モジュール別に関数をソートするには（アルファベット順）、「モジュール」列ヘッダをクリックします。

補足：「モジュール」列はアプリケーションに再配置可能なモジュールがある場合にのみ便利です。

6.2.5 オペコード列の見出し

デフォルトでは、「オペコード」列は非表示ですが（横スペース節約のため）、「関数」ツールバーの「オペコード」ボタンをクリックして表示することができます。ボタンを再度クリックすると列が非表示になります。

オペコード列は、各関数内のアセンブリオペコードの数（関数サイズ）を表示します。アセンブリオペコード数の降順に全関数を並べ替えるときは、「オペコード」列ヘッダをクリックします。

6.2.6 すべて選択解除ボタン

すべての選択フレームまたは選択期間をすばやく解除するには、「すべて選択解除」ボタンをクリックします。

6.2.7 上位選択ボタン

すばやく上位 45 件の関数を選択するには、「関数」ツールバーの「上位選択」ボタンをクリックします。

6.2.8 引数ボタン

デフォルトでは、関数名が「関数」タブと「コールツリー」タブで表示される際に、関数の引数のリストは非表示となっています（画面の見やすさと GUI 描画スピードアップのため）。引数を表示したい場合は、「引数」ボタンをクリックします。

6.2.9 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

6.2.10 選択項目のみボタン

「選択項目のみ」ボタンは現在選択されている項目のみをリストに表示します。このモード有効時に選択解除された項目は表示リストから削除されます。

6.3 関数のフィルタリング

「関数」タブの下の「フィルタ」テキストボックスを使用して関数をフィルタリングができます。これにより特定の関数を検索したり、または関数のクラスや特定の名前空間（GX2 など）を表示させることができます。

テキストボックスに何らかの文字列を入力すると、その文字列を含むアイテムを大文字小文字を同一視して検索します。検索文字列に一致するものだけが表示されます。フィルタリング結果を反転する（そのフィルタとマッチしないものすべてを表示する）には、テキストボックスの右側の感嘆符ボタンをクリックします。一致するアイテムがない場合、テキストボックスがオレンジ色にハイライトされます。

フィルタリング結果を向上するために、最初の左括弧（を含む）までの関数名のみが考慮されます。関数引数の検索一致の表示結果は通常不適切で不必要の場合が多いため、この方法で関数引数内の一致結果を取り除くことができます。

便利なフィルターは以下のとおりです。

表 6-1：便利な簡易フィルター

フィルターテキスト	!ボタン	説明
`(`	N	C++ 関数をすべて表示。
`(`	Y	C 関数をすべて表示。
`;`	N	他の関数とコードを共有する関数をすべて表示。

また正規表現にも対応しています（.NET RegEx 利用、Perl 5 互換）。フィルターで文字を正規表現として解釈するためには、テキストボックス右側の「RegEx」ボタンをクリックしてください。また基本的なフィルターとは異なり、正規表現では大文字・小文字の区別を行うことに注意してください。

便利な正規表現は以下のとおりです。

表 6-2：便利な正規表現フィルター

フィルターテキスト	!ボタン	説明
`^GX2`	N	「GX2」で始まる関数をすべて表示。
`^GX2`	Y	「GX2」以外で始まる関数をすべて表示。
`^GX2|^__`	Y	（「GX2」または「__」以外で始まる）複数の関数を削除。

フィルターテキスト	!ボタン	説明
`(?i)^cos|^sin|^tan`	N	三角関数をすべて表示(すべてにおいて大文字・小文字を区別しない)。
`(?i)a(?-i)nim`	N	「anim」または「Anim」のどちらかを含む関数を表示。
`(?i)math.*vec`	N	「math」の後に「vec」を含む関数を表示(大文字・小文字を区別しない)。
`(?<!trans)pose`	N	「trans」が前に付かない「pose」を含む関数を表示。
`skin(?!g)`	N	「g」が後に続かない「skin」を含む関数を表示。

補足：「関数」、「コールツリー」、および「コードカバレッジ」タブ内のフィルターすべては、関数に対する動作が同様で、保存したフィルターも共有します。

6.3.1 正規表現クイックガイド

正規表現はテキストの文字列を簡潔にマッチさせる方法です。正規表現を指定するときに便利ないくつかのルールは以下になります。

- 正規表現に空白は入れません(空白の検索時を除く)。
- 縦棒はブール演算子 “or” を表します。例えば `cos|sin` は cos または sin のどちらかとマッチします。
- (?i) を使用すると大文字と小文字を区別しません。これを使用するとその後に続く文字列すべてに影響します。例えば `(?i)cos|sin` は大文字・小文字を区別せずに cos または sin のいずれかとマッチします。
- 脱字記号キャレットは関数の先頭部分のみを示します。例えば `^GX2` は GX2 で始まる関数とマッチします。
- ピリオドは任意の文字にマッチします。アスタリスクは 0 個以上の左隣の要素を指定します。この両者を利用して 0 個以上の任意の文字を指定することが可能です。例えば、`math.*vec` は math と vec の間に 0 個以上の文字がある文字列にマッチします。

6.3.2 保存済みフィルター

保存フィルターはプロファイルセッション間で同じフィルターを使用可能にします。プロファイルが開いている時に保存フィルターを読み込みます。デフォルトでは、「フィルタ」ドロップダウンボックスでカスタマイズ可能なフィルター例が表示されます。ドロップダウンボックスに新たなフィルタを追加する場合は、フィルタ文字列を入力してから「保存」(「+」)ボタンを押します(「反転」と「RegEx」ボタンの状態もフィルタ文字列と共に保存されます)。ドロップダウンボックスからフィルタを削除する場合は、まずそのフィルタを選択してから「削除」(「-」)ボタンを押します。

補足：「関数」タブで保存したフィルターは「コールツリー」タブと「コードカバレッジ」タブからも利用できます。ただし「スレッド」タブと「カウンタ」タブの「フィルタ」ロップダウンボックスは共有されないことに注意してください。これはスレッド名またはカウンタ名を検索するタブであり、関数の検索には使用されないためです。

6.4 関数の選択

関数はクリックして選択し、再度クリックして選択を解除します。関数は最大 500 件まで選択できます(選択された関数はそれぞれが違う色でハイライトされます)。選択された関数は「サンプルグラフ」でグラフ表示され、「コールツリー」でハイライトされます(「コールスタック」プロファイルデータでプロファイルを取得時のみ「コールツリー」を表示します。詳細はコールスタック録音オプションを参照)。

すばやく上位 50 件の関数を選択するには、「関数」ツールバーの「上位選択」ボタンをクリックします。

すべての関数をすばやく解除するには、「関数」ツールバーの「すべて選択解除」ボタンをクリックします。

6.4.1 その他の選択方法

Shift キー + クリックで複数アイテムを選択／解除することができます（Shift キーを押さずにアイテムを 1 つ選択/解除してから、Shift キーを押したまま次のアイテムをクリックします）。これにより 2 回のクリック間のアイテムをまとめて選択／解除できます。

アイテムを 1 つだけ選択したまま残りを解除するときは、Alt キー + クリックを使用します。Alt キー + クリックですべてのアイテムの選択解除を行ってから、クリックされたアイテムを選択します。

6.5 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

6.5.1 関数名をクリップボードにコピー

引数付きの関数フルネームをクリップボードにコピーします。

6.5.2 他のすべての関数の選択解除

クリックされている関数以外の関数すべての選択を解除します。未選択関数が選択状態に切り替わることはありません。

6.5.3 アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブに焦点を移動して、クリックした関数のアセンブリを表示します。

6.5.4 詳細表示

「詳細表示」オプションは新規の関数詳細ウィンドウを開いて、クリックした関数の詳細を表示します。

6.6 単位

各アイテムの前に表示される単位を変更するには、リボンバーの「ホーム」タブにある「単位」ボタンを使用します（単位を参照）。

6.7 ウィンドウのサイズ変更

左タブグループのタブすべて、つまり「関数」、「スレッド」、「計装」、および「カウンタ」タブは、右端をマウスカーソルでドラッグすると横方向にストレッチできます。すべてのタブで同じ表示領域を共有するため、あるタブの幅サイズを変更すると、タブグループ内の他のタブのサイズも変更されます。

6.8 隠された関数の名前を確認する

関数の名前が非常に長い場合、その一部が「関数」タブの右端に隠れてしまう場合があります。その場合は「関数」タブのサイズを変更するか、関数にポインタを置いてツールチップ表示すると名前全部を確認できます。こ

のツールチップウィンドウには、「引数」ボタンの選択状態とは関係なく、引数も含めた関数シグネチャの全情報が表示されます。

6.9 1 行中の複数関数名

一部の行では関数名が連結表示されていることがあります。その場合は各関数名はセミコロンで区分されます(「引数」ボタン選択時か、関数上部にポインタを置いてツールチップを表示した場合のみに表示されます)。例えば、「関数」タブの 1 行に以下のように関数が連続表示されていることがあります。

```
function_name_1; function_name_2
```

これは異なる関数がまったく同じコードにコンパイルされている場合に発生します。プロファイラがこれらの共用コード内の関数の 1 つを実行してサンプルを取得した場合、プロファイラでは呼び出し関数を判定できないため、同じ行にすべての関数名を並べて表示します。

6.10 システムアイドルスレッド

プロファイルが複数スレッドを含んでいる場合に、アクティブなスレッドがないときがあれば、その時間は「システムアイドルスレッド」になります。実際には特定のスレッドはシステムコールなどで待機になりますが、プロファイラではどのスレッドが待機しているのかを特定できません。

6.11 未解決の関数

名前が未解決の関数がプロファイルに存在する場合は、その関数を未解決内部関数として分類します。これは通常 `filename.ext_internal (unresolved)` という形式をとり、「`filename.ext`」の部分は、関数のシンボルを読み込んだファイルの名前です。

7 最後タブ

「最後」タブでは、最後にサンプリングしたプロファイルのすべての関数を表示します。これは比較目的のために有益なだけでなく、計装する新規関数を選択するためにも非常に便利です。一旦「計装プロファイル」を取得したら、通常の「関数」タブはもはやデータを含まないため非表示になるからです。「最後」タブ内の関数を選択することはできませんが、「関数」タブ内で操作できるほとんどの関数、例えばすべてのソートおよびフィルター機能の関数はこのタブでも作動します。「最後」タブの関数データ用のユニットは割合とサンプルのみに制限されているので注意してください。

8 スレッドタブ

「スレッド」タブにはプロファイル中にサンプリングしたスレッドがすべて表示されます。

8.1 スレッドの並び替え

それぞれのヘッダをクリックして、スレッドを「セルフ」、「コア」、「名前」でソートできます。デフォルトでは、スレッドは「コア」順、さらに各コア内で「セルフ」順にソートされています。

8.2 スレッドタブのツールバー

「スレッド」タブのツールバー内の操作を以下に示します。

8.2.1 コアのフィルタリングと列見出し

プロファイルに複数のコアが含まれる場合は、ツールバーの「コア」ドロップダウンをクリックして、表示されたリストの中から「コアすべて」やコア別の選択ができます。ただし「コア」ドロップダウンに表示されるのはプロファイル済みのコアのみになります。デフォルトでは「コアすべて」が選択されており、新規のプロファイルを取得したり読み込んだときは、「コア」ドロップダウンメニューがリセットされてデフォルトの「コアすべて」に戻ります。

例えば「コア 1」などの特定のコアを選択すると、表示される関数はすべて同じコアになるため、「コア」列は非表示となります。このオプションは特定のコアに焦点を当てる場合や、タブの横スペースの有効活用に利用できます。

8.2.2 すべて選択解除ボタン

すべての選択フレームまたは選択期間をすばやく解除するには、「すべて選択解除」ボタンをクリックします。

8.2.3 上位ボタン

「トップ」ボタンを押すと、現在のソート設定に基づき、一覧の上位 45 件を選択します。

8.2.4 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

8.3 スレッドの選択

スレッドはクリックして選択し、再度クリックすると選択が解除されます。スレッドは最大 45 件まで選択できます（選択されたスレッドはそれぞれが違う色でハイライトされます）。選択したスレッドは「サンプルグラフ」タブでグラフ表示されます。

すばやく上位 45 件のスレッドを選択するには、「スレッド」ツールバーの「トップ」ボタンをクリックします。

すべてのスレッドをすばやく解除するには、「スレッド」ツールバーの「すべて選択解除」ボタンをクリックします。

8.3.1 その他の選択方法

Shift キー + クリックで複数アイテムを選択／解除することができます（Shift キーを押さずにアイテムを 1 つ選択/解除してから、Shift キーを押したまま次のアイテムをクリックします）。これにより 2 回のクリック間のアイテムをまとめて選択／解除できます。

アイテムを 1 つだけ選択したまま残りを解除するときは、Alt キー + クリックを使用します。Alt キー + クリックですべてのアイテムの選択解除を行ってから、クリックされたアイテムを選択します。

8.4 スレッドのフィルタリング

「スレッド」タブの下の「フィルタ」テキストボックスを使用してスレッドをフィルタリングできます。これにより多数のスレッドが表示されている場合に、その中から特定のスレッドを検索しやすくなります。ただし通常のスレッド数はさほど多くないため、この操作よりもむしろ「スレッド」ツールバーの「コア」ドロップダウンボックスを使用したフィルタリングを推奨します。

テキストボックスに何らかの文字列を入力すると、その文字列を含むアイテムを大文字小文字を同一視して検索します。検索文字列に一致するものだけが表示されます。フィルタリング結果を反転する（そのフィルタとマッチしないものすべてを表示する）には、テキストボックスの右側の感嘆符ボタンをクリックします。一致するアイテムがない場合、テキストボックスがオレンジ色にハイライトされます。

また正規表現にも対応しています（.NET RegEx 利用、Perl 5 互換）。フィルターで文字を正規表現として解釈するためには、テキストボックス右側の「RegEx」ボタンをクリックしてください。また基本的なフィルターとは異なり、正規表現では大文字・小文字の区別を行うことに注意してください。

正規表現の詳細については正規表現クイックガイドを参照してください。

8.4.1 保存済みフィルター

保存フィルターはプロファイルセッション間で同じフィルターを使用可能にします。プロファイルが開いている時に保存フィルターを読み込みます。デフォルトでは、「フィルタ」ドロップダウンボックスでカスタマイズ可能なフィルター例が表示されます。ドロップダウンボックスに新たなフィルタを追加する場合は、フィルタ文字列を入力してから「保存」（「+」）ボタンを押します（「反転」と「RegEx」ボタンの状態もフィルタ文字列と共に保存されます）。ドロップダウンボックスからフィルタを削除する場合は、まずそのフィルタを選択してから「削除」（「-」）ボタンを押します。

> 補足：「スレッド」タブの保存フィルターはその他のフィルターとは独立して保存されます。

8.5 ウィンドウのサイズ変更

左タブグループのタブすべて、つまり「関数」、「スレッド」、「計装」、および「カウンタ」タブは、右端をマウスカーソルでドラッグすると横方向にストレッチできます。すべてのタブで同じ表示領域を共有するため、あるタブの幅サイズを変更すると、タブグループ内の他のタブのサイズも変更されます。

9 計装タブ

「計装」タブにはプロファイル中に記録された計装関数や計装コードブロックがすべて表示されます。ただし「計装」タブが表示されるのは上記のいずれかのデータが記録された場合のみです。

9.1 計装タブのツールバー

「計装」タブのツールバー内の操作を以下に示します。

9.1.1 すべて選択解除ボタン

すべての選択フレームまたは選択期間をすばやく解除するには、「すべて選択解除」ボタンをクリックします。

9.1.2 すべて展開ボタン

「すべて展開」ボタンを押すと表示内で各アイテムを展開します。

9.1.3 すべて非表示ボタン

「すべて非表示」ボタンを押すと表示内で各アイテムを非表示にします。

9.1.4 類似項目の選択ボタン

「類似項目の選択」ボタンが有効になると、アイテムを選択、選択解除すると類似したアイテムも選択、選択解除されます。

「計装」タブの「類似項目の選択」ボタンを押すと、計装およびコードブロック内の類似値の選択が可能になります。「類似項目の選択」では展開されているブロックのみが対象になります。例えば、あるブロックの呼び出し回数をクリックするとすべての展開ブロックに対する呼び出し回数が選択（選択解除）されます。

9.1.5 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

9.2 計装項目

計装項目は展開表示して、プロファイル時に記録したすべてのデータを確認することができます。各計装項目において、以下の数値が表示されます。

- 関数呼び出しまたはブロック：プロファイル中に入力および終了された関数またはコードブロックの回数です。もしこれらの数字が一致しない場合は、どちらかの小さい数字が使用されます（例えば、再帰関数でサンプリングが停止するなど）。
- 最大再帰深度：プロファイル中に検出された再帰の最大深度です。値がゼロ以上であった場合にのみ測定値が表示されます。

- 呼び出しごとの時間またはブロックごとの時間:呼び出しごとまたはブロックのごとの測定平均時間です。(平均時間 = 累計時間 / 呼び出し回数)
- パフォーマンスカウンタデータ:プロファイル時に選択されたパフォーマンスカウンタに基づく計測データです。プロファイル実行中の通算計測値となります。この測定値を呼び出し回数(パーセント値以外の値)で割ると、さらに有益な数値となります。

9.3 グラフ表示する計装項目の選択

各計装項目内でデータフィールドを選択すると、「サンプルグラフ」内で選択データフィールドがグラフ表示されます。データフィールドは一度に最大 45 項目まで選択できます。

展開表示した項目内から類似したデータフィールドを選択するには、まず「計装」タブツールバーで「類似項目の選択」ボタンを有効にしてからデータフィールドを 1 つ選択します。

9.3.1 その他の選択方法

Shift キー + クリックで複数アイテムを選択/解除することができます(Shift キーを押さずにアイテムを 1 つ選択/解除してから、Shift キーを押したまま次のアイテムをクリックします)。これにより 2 回のクリック間のアイテムをまとめて選択/解除できます。

アイテムを 1 つだけ選択したまま残りを解除するときは、Alt キー + クリックを使用します。Alt キー + クリックですべてのアイテムの選択解除を行ってから、クリックされたアイテムを選択します。

9.4 プロファイラのオーバーヘッド

1 回に計装できる関数が 1 つだけという仕様は、プロファイラのオーバーヘッドを考慮した上での意図的なものです。計装対象関数の開始、終了が記録されるごとに、150 から 300 マイクロ秒のオーバーヘッドが発生します(そのうち約半分が「時間」および「サイクル」の結果に含まれます)。

各関数とコードブロック計装内で発生するオーバーヘッドをよりわかりやすく確認するために、計装プロファイルには「ゼロサイズブロック(ブロックオーバーヘッドごと)」というコードブロックが含まれます。オーバーヘッドの調査やグラフ化にこれを利用できます。このコードブロックはアプリケーションが実際に呼び出したもので、それ自体には内部処理を持ちません。この結果をもとに、他の計装した関数およびコードブロックの判断を調整してください。

9.5 ウィンドウのサイズ変更

左タブグループのタブすべて、つまり「関数」、「スレッド」、「計装」、および「カウンタ」タブは、右端をマウスカーソルでドラッグすると横方向にストレッチできます。すべてのタブで同じ表示領域を共有するため、あるタブの幅サイズを変更すると、タブグループ内の他のタブのサイズも変更されます。

10 カウンタタブ

「カウンタ」タブはプロファイル中にサンプリングしたカウンタをすべて表示します。

10.1 カウンタの種類

現在プロファイラは以下の一般カウンタに対応しています。各プラットフォームにはそれ以外のカウンタも用意されています。

10.1.1 パフォーマンスカウンタ

パフォーマンスカウンタの詳細はパフォーマンスカウンタグループを参照してください。

10.2 カウンタのソート

「コア」、「グループ」または「名前」のそれぞれの見出しをクリックして、カウンタをソートできます。デフォルトは、カウンタはコア別でソートされています。

10.3 カウンタタブのツールバー

「カウンタ」タブのツールバー内の操作は以下のとおりです。

10.3.1 コアのフィルタリングと列見出し

プロファイルに複数のコアが含まれる場合は、ツールバーの「コア」ドロップダウンをクリックして、表示されたリストの中から「コアすべて」やコア別の選択ができます。ただし「コア」ドロップダウンに表示されるのはプロファイル済みのコアのみになります。デフォルトでは「コアすべて」が選択されており、新規のプロファイルを取得したり読み込んだときは、「コア」ドロップダウンメニューがリセットされてデフォルトの「コアすべて」に戻ります。

例えば「コア 1」などの特定のコアを選択すると、表示される関数はすべて同じコアになるため、「コア」列は非表示となります。このオプションは特定のコアに焦点を当てる場合や、タブの横スペースの有効活用に利用できます。

10.3.2 グループフィルタリングと列見出し

「カウンタ」タブにある「グループ」ドロップダウンを使用して、「全グループ」とプロファイル内にあるいずれかのグループの間で選択を切り替えることが可能です（例えば、パフォーマンスカウンタ）。ただし「グループ」ドロップダウンに表示されるのはプロファイル内のグループのみになります。デフォルトでは「全グループ」が選択されており、新規のプロファイルの取得や読み込み時には、「グループ」ドロップダウンメニューがリセットされて「全グループ」に戻ります。

例えば「パフォーマンスカウンタ」などの特定のグループを選択すると、表示されるグループはすべて同じ種類になるため、「グループ」列は非表示となります。

とりわけパフォーマンスカウンタグループにおいては、ドロップダウンにプロファイルで記録したパフォーマンスカウンタグループの追加情報が表示されます（パフォーマンスカウンタグループを参照）。例えば、パフォーマンスカウンタグループの「グループ例」をプロファイルした場合は、「グループ」ドロップダウンには「パフォーマンスカウンタ（グループ例）」の入力情報が表示されます。

10.3.3 すべて選択解除ボタン

すべての選択フレームまたは選択期間をすばやく解除するには、「すべて選択解除」ボタンをクリックします。

10.3.4 類似項目の選択ボタン

「類似項目の選択」ボタンが有効になると、アイテムを選択、選択解除すると類似したアイテムも選択、選択解除されます。

「カウンタ」タブの「類似項目の選択」ボタンを押すと各コアにおける類似カウンタを選択できます。例えば、パフォーマンスカウンタグループをプロファイルした場合、コア 0 の 1 番目のカウンタをクリックするとプロファイルしたコアすべての 1 番目のカウンタの選択・選択解除を行います。

10.3.5 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

10.4 カウンタのフィルタリング

テキストボックスに何らかの文字列を入力すると、その文字列を含むアイテムを大文字小文字を同一視して検索します。検索文字列に一致するものだけが表示されます。フィルタリング結果を反転する(そのフィルタとマッチしないものすべてを表示する)には、テキストボックスの右側の感嘆符ボタンをクリックします。一致するアイテムがない場合、テキストボックスがオレンジ色にハイライトされます。

また正規表現にも対応しています(.NET RegEx 利用、Perl 5 互換)。フィルターで文字を正規表現として解釈するためには、テキストボックス右側の「RegEx」ボタンをクリックしてください。また基本的なフィルターとは異なり、正規表現では大文字・小文字の区別を行うことに注意してください。

正規表現の詳細については正規表現クイックガイドを参照してください。

10.4.1 保存済みフィルター

保存フィルターはプロファイルセッション間で同じフィルターを使用可能にします。プロファイルが開いている時に保存フィルターを読み込みます。デフォルトでは、「フィルタ」ドロップダウンボックスでカスタマイズ可能なフィルター例が表示されます。ドロップダウンボックスに新たなフィルタを追加する場合は、フィルタ文字列を入力してから「保存」(「+」)ボタンを押します(「反転」と「RegEx」ボタンの状態もフィルタ文字列と共に保存されます)。ドロップダウンボックスからフィルタを削除する場合は、まずそのフィルタを選択してから「削除」(「-」)ボタンを押します。

> 補足：「カウンタ」タブの保存フィルターはその他のフィルターとは独立して保存されます。

10.5 カウンタの選択

カウンタをクリックして選択し、再度クリックして選択を解除します。カウンタは最大 45 件まで選択できます(選択されたカウンタはそれぞれ違う色でハイライトされます)。選択したカウンタは「サンプルグラフ」タブでグラフ表示されます。

すべてのカウンタを解除するには、「すべて選択解除」ボタンをクリックします。

10.5.1 その他の選択方法

Shift キー + クリックで複数アイテムを選択／解除することができます（Shift キーを押さずにアイテムを 1 つ選択/解除してから、Shift キーを押したまま次のアイテムをクリックします）。これにより 2 回のクリック間のアイテムをまとめて選択／解除できます。

アイテムを 1 つだけ選択したまま残りを解除するときは、Alt キー + クリックを使用します。Alt キー + クリックですべてのアイテムの選択解除を行ってから、クリックされたアイテムを選択します。

10.6 ウィンドウのサイズ変更

左タブグループのタブすべて、つまり「関数」、「スレッド」、「計装」、および「カウンタ」タブは、右端をマウスカーソルでドラッグすると横方向にストレッチできます。すべてのタブで同じ表示領域を共有するため、あるタブの幅サイズを変更すると、タブグループ内の他のタブのサイズも変更されます。

10.7 単位

各アイテムの前に表示される単位を変更するには、リボンバーの「ホーム」タブにある「単位」ボタンを使用します（単位を参照）。

11 情報タブ

「情報」タブはプロファイルの一般情報を表示します。

11.1 情報タブのツールバー

「情報」タブのツールバー内の操作は以下のとおりです。

11.1.1 すべて展開ボタン

「すべて展開」ボタンを押すと表示内で各アイテムを展開します。

11.1.2 すべて非表示ボタン

「すべて非表示」ボタンを押すと表示内で各アイテムを非表示にします。

11.1.3 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

11.2 環境

「環境」セクションにはプロファイル時のプロファイラ操作環境に関する情報が表示されます。以下の情報が表示されます。

- ユーザー:コンピューターにログインしているユーザー名。
- コンピューター:プロファイラを実行しているコンピューター名。
- 時間:プロファイル取得時間。
- ゲーム:ハードドライブの実行ファイルディレクトリ。
- キャプチャに使用する機材:プロファイルのキャプチャに使用する機材の名前とタイプ。

11.3 サンプリング設定

「サンプリング設定」セクションはプロファイル作成に使用する設定を表示します。設定内容はコア別に一覧表示されます。

- サンプリングリクエストのレート、時間別またはパフォーマンスカウンタイベントトリガ別。
- コールスタックプロファイルの有効、無効。
- プロファイルでパフォーマンスカウンタの記録の有無。記録する場合は選択するパフォーマンスカウンタグループの種類。
- フレームマーカーがコア上で確認されたかどうかの有無。

出力例

```
Core 0：10x per frame, Callstack Sampling, Has frame markers
```

11.4 プロファイル統計

「プロファイル統計」タブはプロファイルの一般情報を表示します。表示内容は以下になります。

- 使用バッファ：生データバッファサイズ。
- コア別の表示内容
  - サンプル：コア上のサンプル回数。
  - 持続時間：コア上のプロファイル持続時間。コア上のプロファイル時間には多少の差異が発生することもあります。
  - 実行関数：コア上で実行が確認された関数の数。
  - 実行スレッド：コア上で実行が確認されたスレッドの数。
  - 最大コールスタック深度：プロファイル中のコールスタックの最大深度。（コールスタックプロファイル取得時のみ表示。）
  - コールツリーノード：「コールツリー」タブの合計ノード数。（コールスタックプロファイル取得時のみ表示。）

11.5 モジュール統計

「モジュール統計」セクションは、プロファイル中に確認された各種モジュールを表示します。モジュールは「記録モジュール」と「解決モジュール」に分類されます。

「記録モジュール」はプロファイラでロードやアンロードが確認された再配置可能モジュールです。各記録モジュールは解決モジュールリストに表示されます。

「解決モジュール」は実際にプロファイラにロードされたモジュールで、シンボルアドレスやシンボル名の解決に使用します。この常駐モジュールは解決モジュールリストに表示されますが、記録モジュールリストには表示されません。

各モジュール一覧には以下の情報が表示されます。

- モジュール名。
- モジュールロード元のフォルダ（「解決モジュール」のみ）。
- ベース：モジュールをメモリへロードするアドレス。
- サイズ：メモリ内のモジュールサイズ（「記録モジュール」のみ）。
- 関数：モジュールからロードされた関数の数（「解決モジュール」のみ）。
- 寿命：モジュールがメモリに読み込まれた時間情報。フォーマットは [time loaded, time unloaded] になります。

11.6 推測ハートビート

「推測ハートビート」セクションは、プロファイルデータ分析中に確認された推測ハートビートを表示します。以下の情報が推測ハートビートごとに表示されます。

- 推測ハートビートが検出されたコア。
- 期間：関数実行期間。
- 変化：関数がどの程度定期的に実行されているか。
- スレッド：関数を実行しているスレッドの ID。

- 関数名。

11.7 アプリケーション情報

「アプリケーション情報」は実行アプリケーションに関する一般データを表示します。記録アイテムは以下のとおりです。

- 平均フレームレート:プロファイル中に検出された平均フレームレートです。値を取得するためには、フレームマーキングが有効になっている必要があります。
- ビルド **SDK**:アプリケーションビルドに使用された SDK。
- ビルドタイプ:ビルドタイプ(デバッグまたはリリース)。

11.8 プロファイラ情報

プロファイラ統計セクションは、プロファイル内データの記録に使用されたプロファイラランタイム情報を表示します。これらの値は直接便利な情報となることはあまりないですが、プロファイラで確認された問題の報告用に便利といえます。

すべてのニンテンドー 3DS プロファイルで以下の情報を確認できます。

- プロファイラヘッダバージョン:プロファイラ内で使用される内部マーカーで各種設定を示します。

以下は最新のプロファイラに追加された項目になります。

- プロファイラバージョン:データを記録するプロファイラのバージョンです。
- プロファイラビルド **SDK**:プロファイラビルドに使用される SDK です。

11.9 補足事項

「情報」タブの下にある「備考」テキストボックスにはプロファイルと共にユーザーメモを追加できます。この備考は「保存」ボタンでプロファイル保存時に一緒に保存されます。

備考が空欄の場合は、「備考」セクションのヘッダのみが表示されてセクション自体は非表示になります。「備考」セクションを展開するときは、「備考」のヘッダ右側の上矢印をクリックするか、「備考」のヘッダをマウスで上方向にドラッグします。

12 チェッカータブ

「チェッカー」タブではプロファイルに対して実行したチェッカーの一覧が表示され、潜在的な問題の検出を行います。

新規プロファイルを開発キットから受信したり、プロファイルを読み込むと、チェッカー機能がプロファイルに対して自動的に実行されます。チェッカー処理が完了すると、問題を検出したチェッカーはその結果をタブ上で表示します。問題を検出しなかったチェッカーは表示されません。

12.1 チェッカータブのツールバー

「チェッカー」タブのツールバー内の操作は以下のとおりです。

12.1.1 チェッカー管理

「チェッカー管理」ボタンをクリックすると「チェッカー」タブのチェッカーすべてが表示されます。一覧には無効のチェッカーや結果が未取得のチェッカーも含まれます。本機能には、無効なチェッカーを再び有効にするだけでなく、結果が取得されなかったチェッカーのパラメータの修正も可能です。

この管理モードでは、チェッカーの状態が各チェッカータイトルの一番右側に表示されます。一般的にチェッカーの標準実行時には結果が表示されますが、特別なラベルは右側に表示されません。有効なチェッカーが結果を取得しなかった場合、「結果なし」のテキストが表示されます。チェッカーが無効の場合は、「チェッカー無効」のテキストが表示されます。

> 補足： 無効なチェッカーは管理モードでは起動しません。チェッカー機能の結果を確認するには、有効にする必要があります。

12.1.2 ステータス

「チェッカー管理」ボタン右のステータス領域には、起動中チェッカーの現在のステータスが表示されます。

表示されるステータスタイプは 2 種類あります。

- **ELF 処理終了待ち**:ELF 上でバックグラウンド処理が行われていることを示します。一般的に、各関数コード内の静的関数呼び出しを探し出す、ゲームの静的分析に起因するものです。
- チェッカー実行中 ***checker_name***:チェッカーがプロファイルデータ上で現在実行中であることを示します。

12.2 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

各チェッカーのコンテキストメニューは、チェッカー結果表に保存される値に基づいて生成されます。各チェッカーのコンテキストメニューには、以下の項目のうち該当する項目が表示されます。

チェッカー結果内の特定項目が当該メニュー操作に対応していない場合は、コンテキストメニューの項目は無効となります。

12.2.1 関数の選択

「関数の選択」オプションは、クリックした関数を「関数」タブ上で選択状態にします。

12.2.2 詳細表示

「詳細表示」オプションは新規の関数詳細ウィンドウを開いて、クリックした関数の詳細を表示します。

12.2.3 アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブに焦点を移動して、クリックした関数のアセンブリを表示します。

12.2.4 フレーム単位負荷の分析

「フレーム単位負荷の分析」オプションは、この結果入力に対する対象関数すべて選択して、「サンプルグラフ」内のグループ全体に対するフレーム単位負荷を表示します。またこのオプションを使用すると、「サンプルグラフ」タブが表示されます。

表示をリセットするには、分析タブの「リセット」ボタンをクリックします。

12.3 重要度

各チェッカーの重要度は、リストのアイコンおよび位置で特定可能です。重要度が最も高い項目はリストの一番上に表示されます。

- 紫の山形マークの項目は重要度が最も高く、深刻な問題。
- 赤い四角マークの項目は重要度が高く、深刻な問題。
- オレンジのひしがたマークの項目は重要な問題を示しており、更なる調査が必要。
- 黄色の三角マークの項目は重要度が低い問題を示しており、それほど深刻ではない。
- 緑の丸マークの項目は情報提供のみ。

12.4 チェッカー結果

チェッカーが報告対象の項目を検出すると、その結果が「チェッカー」タブに追加されます。各エントリには特定アイテムセットが含まれます。

12.4.1 チェッカー再実行ボタン

「チェッカー再実行」ボタンを押すと、現在の設定でチェッカーを再実行します。このときリスト上のチェッカーの結果は削除し、結果を新規に取得後、この新規結果をリストに追加して表示します。

12.4.2 テキストのコピーボタン

「テキストのコピー」ボタンを押すと、「概要」、「説明」および「修正案」をクリップボードにコピーできます。

12.4.3 表のコピーボタン

「表のコピー」ボタンをクリックすると、「結果表」をクリップボードにコピーできます。コピーされたテキストにはタブ区切りされた値が含まれているので、スプレッドシートアプリケーションに直接ペーストすることが可能です。

12.4.4 チェッカー無効ボタン

「チェッカー無効」ボタンを押すと、チェッカー機能の「チェッカー」タブ内での起動や表示が無効になります。チェッカー機能の有効 / 無効状態は設定に保存されるため、複数のプロファイルが実行されても設定は持続します。

チェッカー機能を再び有効にするには、「チェッカー」タブツールバーのチェッカー管理ボタンを使用します。

> 補足： チェッカー機能の有効・無効化を切り替えると、対象チェッカーが自動的に再試行されることになります。

12.4.5 概要

すべての検出結果には、簡単な「概要」が表示されています。検出結果がない場合も、その旨が「概要」に表示されます。

チェッカーの実行に 1 秒以上かかる場合は、チェッカー処理時間が「概要」の下部に表示されます。

12.4.6 設定可能なパラメータ

チェッカーに設定可能な項目がある場合は、スピナー操作が「概要」の下部に表示されます。各スピナーにはまずデフォルト値が表示されて、入力可能な最大および最小値が設定されています。特定のアプリケーションごとにパラメータを調整するとより良いチェッカー結果を取得できます。一度これらのパラメータを変更すると、次回プロファイラを実行する際のために保存されます。パラメータをデフォルト値にリセットするには（保存済みの値を上書きする）、「デフォルトにリセット」ボタンをクリックします。

12.4.7 デフォルトにリセットボタン

「デフォルトにリセット」は、チェッカーの設定可能なパラメータ すべてをデフォルト値にすばやくリセットします。

12.4.8 結果表

チェッカーはそれぞれ「結果表」を作成します。チェッカー内で必ず表示される固定列はありません。

チェッカー内で一般的に表示される列は以下になります。

- セルフ：プロファイル結果内で対象関数に起因する実行時間
- 関数：検出関数の名前

12.4.8.1 結果表の並び替え

各「結果表」の並び替え順には、チェッカー結果に対してもっとも一般的に有用な並び替え順であると判断されたデフォルト設定が使用されています。

もしこのデフォルト並び替え順が適当でない場合、または結果を別の方法で表示したい場合は、結果内で使用する並び替えを変更できます。表内のヘッダーのいずれかをクリックして、その列別に結果を並び替えます。この操作を行うときに列内に表示される矢印で、並び替え対象列および並び替え方向が確認できます。同じ列を再度クリックすると、並び替え方向が変更されます。

12.4.9 説明

「説明」の詳細では、チェッカーの検証対象項目およびその検証理由を説明しています。

12.4.10 修正案

「修正案」の詳細では、チェッカーによって報告された問題修正のための、アプリケーション修正案を表示します。

13 コールツリータブ

このタブではプロファイルしたサンプルデータに基づき検出したコールツリーを表示します。このタブは「コールスタック」データをオンにしてプロファイルを取得した場合にのみ表示されます(コールスタック録音オプション参照)。

13.1 推測コールツリー

「推測コールスタック」を取得した際、このタブは「コールツリー」から「推測コールツリー」に変わります。

13.2 コールツリータブ

「コールツリー」タブのツールバー内の操作は以下のとおりです。

13.2.1 プレフィックス選択

各関数名の前にある接頭辞は、「コールツリー」ツールバーの 3 種類の「プレフィックス」トグルボタンで決定します。デフォルトで「合計」および「セルフ」プレフィックスがオンになっています。プレフィックスのオン、オフを切り替えるには、各プレフィックスボタンをクリックします。すべてのボタンがオフの場合は、関数の名前のみを表示します。プレフィックスの選択肢は以下になります。

合計:各関数の実行時間とその関数が呼び出した関数の実行時間を表示します。

セルフ:各関数が呼び出す関数の実行時間を除いた、その関数のみの実行時間を表示します。

サブ:各関数が呼び出した関数の実行時間を表示します。

ツリーの各レベルの関数はプレフィックスによりソートされていることに注意してください。「合計」、「セルフ」、あるいは「サブ」を選択すると、左端プレフィックスの降順(大きな数字から小さな数字)でソートされます。プレフィックスを選択しない場合は、関数名はアルファベット順でソートされます。

13.2.2 すべて展開ボタン

「すべて展開」ボタンを押すと表示内で各アイテムを展開します。

13.2.3 すべて非表示ボタン

「すべて非表示」ボタンを押すと表示内で各アイテムを非表示にします。

13.2.4 自動展開表示

デフォルトでは、「コールツリー」は自動展開表示を行いませんが「自動展開」ドロップダウンボックスで設定を有効にすることが可能です。自動展開表示を有効にすると、マウスクリックでどの関数が選択されたか、また「検索」機能(「関数」タブまたは「コールツリー」タブ内)の結果でどの関数がハイライトされたかに基づいて、ツリーは自動展開表示・非表示します。

「自動展開」ドロップダウンボックスには3つの選択肢があります。

- 自動展開オフ:関数の選択の有無、または「検索」機能でハイライトされた関数に基づいた「コールツリー」の展開表示、非表示を行いません。
- 自動展開表示のみ:関数の選択の有無、または「検索」機能でハイライトされた関数に基づいた「コールツリー」の展開表示をします(非表示はしません)。

- 自動展開・非表示：関数の選択の有無、または「検索」機能でハイライトされた関数に基づいた「コールツリー」の展開表示、非表示を行います。

さらに「すべて展開」と「すべて非表示」ボタンで、ツリー図全体を簡単に展開、非表示と切り替えできます。

13.2.5 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

13.2.6 全て保存ボタン

「全て保存」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンとは異なり、「すべてコピー」では選択項目の有無は関係ありません。このボタンを使用すると表示中コールツリー全体のコピーが可能です。

13.3 関数の検索

テキストボックスに何らかの文字列を入力すると、その文字列を含むアイテムを大文字小文字を同一視して検索します。検索文字列に一致するものだけが表示されます。フィルタリング結果を反転する（そのフィルタとマッチしないものすべてを表示する）には、テキストボックスの右側の感嘆符ボタンをクリックします。一致するアイテムがない場合、テキストボックスがオレンジ色にハイライトされます。

また正規表現にも対応しています（.NET RegEx 利用、Perl 5 互換）。フィルターで文字を正規表現として解釈するためには、テキストボックス右側の「RegEx」ボタンをクリックしてください。また基本的なフィルターとは異なり、正規表現では大文字・小文字の区別を行うことに注意してください。

便利な関数フィルターは便利な簡易フィルターおよび便利な正規表現フィルター表を確認してください。正規表現の詳細については正規表現クイックガイドを参照してください。

> 補足：ここでの「検索」機能は「関数」や「コードカバレッジ」タブの「フィルタ」操作とは異なります。「コールツリー」表示では、すべての関数が常に表示されます。その代わり、検索一致結果のみがハイライトされます。

13.3.1 保存済みフィルター

保存フィルターはプロファイルセッション間で同じフィルターを使用可能にします。プロファイルが開いている時に保存フィルターを読み込みます。デフォルトでは、「フィルタ」ドロップダウンボックスでカスタマイズ可能なフィルター例が表示されます。ドロップダウンボックスに新たなフィルタを追加する場合は、フィルタ文字列を入力してから「保存」（「+」）ボタンを押します（「反転」と「RegEx」ボタンの状態もフィルタ文字列と共に保存されます）。ドロップダウンボックスからフィルタを削除する場合は、まずそのフィルタを選択してから「削除」（「-」）ボタンを押します。

補足：「関数」タブで保存したフィルターは「コールツリー」タブと「コードカバレッジ」タブからも利用できます。ただし「スレッド」タブと「カウンタ」タブの「フィルタ」ロップダウンボックスは共有されないことに注意してください。これはスレッド名またはカウンタ名を検索するタブであり、関数の検索には使用されないためです。

13.4 関数の選択

「関数」タブで関数を選択する方法と同様に、「コールツリー」内の関数をクリックして関数をハイライトします（最大 45 件までの関数をハイライト可能です）。ハイライトされた関数の全インスタンスが「コールツリー」でもハイライトされます。自動展開表示がオンの場合、展開されたコールツリーが選択関数の全インスタンスを表示します。

13.5 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

13.5.1 手動展開表示

コールツリーは標準 Windows のディレクトリと同様の階層構造で、プラスまたはマイナスの記号をクリックして各ノードツリーを開閉できます。さらにツリー内の 1 つのノードを右クリックして、そのノードを基点として開閉することもできます。

13.5.2 アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブに焦点を移動して、クリックした関数のアセンブリを表示します。

13.5.3 詳細表示

「詳細表示」オプションは新規の関数詳細ウィンドウを開いて、クリックした関数の詳細を表示します。

13.6 リバースコールツリー

関数を選択すると、「コールツリー」タブの下部にある「リバースコールツリー」内にもその関数が表示されます。この表示では選択した関数をルートとし、ルートを拡張していくことによりコールツリー内を順々に下から上に移動していきます。しかし、「リバースコールツリー」は単にコールツリーの逆表示というわけではありません。関数の呼び出し方によってグループ化し、通常では把握が困難な関数の相関関係を明確にします。例えば、ベクトルの正規化関数が「コールツリー」内で 10 回表示される場合でも、実際には異なる 3 つの関数から呼び出されただけという場合があります。この場合、「リバースコールツリー」内ではベクトルの正規化関数は 3 回表示され、各親関数から呼び出された合計時間がそれぞれ表示されます。

「リバースコールツリー」ウィンドウはヘッダを上下にドラッグしてサイズ変更できます。ヘッダの右側にある下矢印を 1 回クリックしてツリーを折りたたんだり、同様に上矢印をクリックして展開できます。

13.7 単位

各アイテムの前に表示される単位を変更するには、リボンバーの「ホーム」タブにある「単位」ボタンを使用します（単位を参照）。

13.8 詳細不明なスタック

サンプルを収集しても、プロファイラがスタックを十分にトラバースできない場合があります。この場合、プロファイラはコールスタックの未検出セクションを「コールツリー」内に「詳細不明なスタック」ノードとしてマークします。このような検出不可能なセクションは、以下の場合に作成される傾向があります。

- スタックの深度が変数の値で変更された場合。
- スタックの深度が 64 KByte を超過している場合。
- 自動フレーム検出機能の使用時に、ヘッダの下に `nngxWaitVSync` が表示される場合。

14　サンプルグラフタブ

「サンプルグラフ」は選択された関数、スレッド、カウンタ、および計装関数やコードブロックなどを視覚的に表現する機能です。それぞれのタブの各項目をクリックして「サンプルグラフ」内に描画します。

「時間によるサンプル」を選択した場合、関数やスレッドの各サンプルは、サンプル開始時点から次のサンプル時点までを矩形で表示します。サンプル時間はそれぞれ異なるため、矩形の幅も変化します。また、同じ関数やスレッドを繰り返しサンプルする場合は、より長い矩形になります。

「サンプリングストラテジー」ドロップダウンボックスから「パフォーマンスカウンタによるサンプル」オプションの 1 つが使用された場合は、関数およびスレッドのサンプルはその記録時点を示す 1 本の縦線で表示されます。「パフォーマンスカウンタによるサンプル」サンプリングでは、時間に対して不規則な割合でサンプルが記録されることがあります（一部のサンプル間にかなりの大小ばらつきが発生します）。パフォーマンスカウンタのイベントは不規則に発生するため、これは正常な動作となります。

ログされたハートビートまたは推測ハートビートが対象コアに対して選択されている場合は、各選択済み関数およびスレッドにおいて「フレーム単位負荷」グラフも描画されます。「フレーム単位負荷」グラフではフレーム単位で対象関数の性質を迅速に評価できます。濃色の線は実際のデータを、ハイライトは許容誤差（グラフ線をたどりながらデータの標準値を示します）を表示します。また、リボンバー上の「ホーム」タブで選択された「単位」は「フレーム単位負荷」グラフの縦軸を操作します。

> 補足：「フレーム単位負荷」グラフの縦軸目盛りは「サンプルグラフ」ツールバーのドロップダウンメニューで操作できます。

カウンタでは、測定値の折れ線グラフを描画します。カウンタの種類で説明しているカウンタにおける測定値の多くは、複数サンプル間で蓄積されサンプル記録後にクリアされます。そのためサンプリング時のカウンタ値はサンプリングされた関数のみに関連付けられています。その一方、コード内部のユーザーがログしたカウンタは瞬時に記録されてそのままグラフ化されます。

計装関数とコードブロックのグラフは、選択されたデータのタイプによって異なります。ただし、関数またはコードブロックの開始から終了までの定数値が記録データとなるため、記録内容はその値の高さとなる矩形で表示されます。矩形の左辺は関数またはコードブロックの開始時間を示し、右辺は終了時間を示します（矩形の長さが関数またはコードブロックの継続時間となります）。例えば計装関数の「コールごとの時間」をグラフ化すると、矩形の高さはその関数呼び出し（およびその関数が呼び出す関数すべて）に要する時間を示し、その矩形の幅も関数の呼び出し時間を表します。計装関数の「関数呼び出し」をグラフ化する場合は、矩形の高さは必ず 1 となり（該当関数の呼び出し回数は 1 度だけとなるため）、幅は持続時間を表します。パフォーマンスカウンタをグラフ化する場合は、矩形の高さは関数呼び出しまたはコードブロックのカウンタ値を示し、幅はその持続時間を示します。

「サンプルグラフ」タブは DirectX 10 を使用して描画されます。「サンプルグラフ」タブで DirectX を初期化できない場合は、エラーの詳細を示すエラーメッセージが画面に表示されます。DirectX 関連のトラブルシューティングはトラブルシューティングで説明しています。

14.1　サンプルグラフタブのツールバー

「カウンタ」タブのツールバー内の操作は以下のとおりです。

14.1.1　マウスモード

「サンプルグラフ」ツールバー左端のメニューは「サンプルグラフ」内のマウスモードを操作します。4 つのマウスモードがあります。

(1) スクロール:左クリック アンド ドラッグで、グラフが上下、左右にスクロール移動します。

修飾キー:

- Ctrl キーを押したままキーを離すまでの間は、一時的に「計測時間」マウスモードに切り替わります。

(2) フレーム選択:左クリック、または左クリック アンド ドラッグで、ハートビートフレームが選択されます。フレームの選択を解除するには、すでに選択済みのフレームを左クリック、または左クリック アンド ドラッグします。

修飾キー:

- フレームの選択解除は左クリック アンド ドラッグする方法に加えて、Alt キーを押したままでも可能です。
- Ctrl キーを押したままキーを離すまでの間は、一時的に「スクロール」マウスモードに切り替わります。

(3) 時間選択:左クリック アンド ドラッグして時間の範囲を選択します。この時間は最初にクリックされたコア内のサンプル境界線にスナップされることに注意してください。時間の選択を解除するには、すでに選択済みの時間を左クリックまたは左クリック アンド ドラッグします。

修飾キー:

- 時間の選択解除は、左クリック アンド ドラッグする方法に加えて、Alt キーを押したままでも可能です。
- Ctrl キーを押したままキーを離すまでの間は、一時的に「スクロール」マウスモードに切り替わります。

(4) 計測時間:左クリック アンド ドラッグで時間の計測範囲を選択します。左マウスボタンを離すと計測範囲が消えてなくなります。

修飾キー:

- Ctrl キーを押したままキーを離すまでの間は、一時的に「スクロール」マウスモードに切り替わります。

「フレーム選択」および「時間選択」モードで範囲を選択すると、「関数」、「スレッド」、「コールツリー」、および「カウンタ」タブ内の値はその選択オプションに基づいて再計算されます。

補足 : すべてのマウスモードで動作するキー操作もあります。

- 左または右矢印キーを使用してページを左右に移動させたり、上下矢印キーを使用してズーム調整が可能です。
- 関数名をダブルクリックすると、その直下の子関数を展開表示できます。このオプションは右クリックコンテキストメニューでも使用できます。

注意 : フレームまたは時間を個別に選択する場合は、統計データの量が大幅に減少するため、誤差が非常に大きくなります。リボンバー上の「ホーム」タブのエラーボタンを有効にして誤差を再確認してください。

14.1.2 すべて選択解除ボタン

すべての選択フレームまたは選択期間をすばやく解除するには、「すべて選択解除」ボタンをクリックします。

14.1.3 アイシクルグラフ

「アイシクルグラフ」は各記録サンプルにおけるコールスタックをビジュアル化したものです。グラフ描画をするには、まずコア上の関数を最低 1 つ選択して、1 つのフレームまたはフレームセットに焦点を当ててから、「アイシクルグラフ」トグルボタンをチェックします。このグラフの情報は非常に詳細なため、1 つのフレームや小さなフレームセットを拡大表示することを推奨します。「アイシクルグラフ」という名称はグラフの表示形式からつけられたもので、上部にルート関数を、先端の下側方向にコールスタックを表示します。

コールスタックをビジュアル化するため、「アイシクルグラフ」では、ルート関数から各コールスタックのリーフへという方向で呼び出し挙動を示します。各関数名や統計値（深度、合計、およびセルフ）を確認するには、希望のセグメントにマウスオーバーします。各関数の色はプロファイラの自動システム分類機能に基づいて設定されます。カラーキーは各コアのヘッダラベル上部に表示されます。自動システム分類機能は一部の関数で正しく動作しない場合もありますが、平均では問題なく機能し、実行中のシステムの確認に便利です。

複数コアに対する「アイシクルグラフ」の描画には時間がかかることもあり、プロファイラのズーム処理やスクロール処理が遅く感じることもあります。描画時間をスピードアップするには、1 つのコアに対して「アイシクルグラフ」は 1 つだけ描画することを推奨します（複数コアで関数、スレッド、カウンタを選択しないこと）。「アイシクルグラフ」は関数、スレッド、またはカウンタが選択されているコアでのみ描画されるため、各選択は 1 つのコアのみに制限しておきます。

フレームを個別に確認する場合は、「フレーム単位負荷」グラフはこのズームレベル設定ではあまり役に立ちません。「フレーム単位負荷」グラフをすばやく非表示にするには、「サンプルグラフ」ツールバーの「負荷グラフ」ボタンの選択を解除します。

14.1.4 アイシクル選択グラフ

「アイシクル選択」グラフは「アイシクルグラフ」に似ている機能です。しかし、各関数を色別に表示する代わりに、「アイシクルグラフ」では現在選択中の関数のみが色表示されます。このグラフでは、コールスタック内で特定関数が特定の時間に呼ばれた箇所を簡単に確認することができます。

14.1.5 スタックグラフ

「スタックグラフ」は、プロファイル全体でスタックデプスを追跡するための迅速かつ容易な方法を提供します。各バーの色はリーフ関数の分類に基づいて設定されます。カラーキーは各コアのヘッダラベル上部に表示されます。

プロファイラがスタックデプスを取得できない場合は、グラフ値は 0 に固定されます。

14.1.6 負荷グラフ

このボタンで、「サンプルグラフ」の「フレーム単位負荷グラフ」の表示および非表示が簡単に操作できます。「フレーム単位負荷」グラフのデプスコントロールに関する詳細は、サンプル、負荷、およびカウンタのスケールを参照してください。

14.1.7 ヒートマップモード

「ヒートマップモード」は、各フレームデータがお互いに重なりあってサンプルをうっすらと描画するため、繰り返される傾向がビジュアル化できます。このモードを最大限に活用するには、サンプリングレートを低く設定します（フレーム毎のサンプル数 1000 以下）。1 つのフレーム内では正確な動作を確認するためのデータが不足します。しかし、サンプルをお互いに重ねることで、何十、何百といったフレームを活用してフレーム内のゲームパフォーマンスをビジュアル化表現できます。

「ヒートマップモード」ではフレームを重ねていくため、ハートビートフレームが選択されていないコアはこのモードでは非表示になります。すべてのコアでハートビートフレーム処理を行うように設定するには、リボンバーの「ヒートマップモード」タブで、任意のコアで希望のハートビートを選択してから、「すべてのコアに適用」をクリックします。

「ヒートマップモード」に表示される黒い縦線はスタックされた各フレームの端を示しています。この線はうっすらと描画されているため、多くのスタックフレームが重なりながら蓄積されていくと濃い色の線になっていきます。描画される線により、各フレームの時間がたいてい同じ長さであるかかどうか（お互いが直接重なりあうことで）、特定の複数フレームレートに集中しているか、またはそれぞれ広範囲にわたって散らばっているかを確認できます。

プロファイルにはさまざまなフレームレートが含まれていることが多いため、「ヒートマップモード」での分析は容易ではありません。そのため、「ハートビート」タブの「グループ選択」オプションを活用します。まずフレームレートを 1 つに絞り(「20Hz」など)、この低いレートのフレームのみを重ねていくことで、無関係のフレームによるノイズを削除します。また「分析」タブの「ストーリー分析」機能を使用して、特定フレーム内の重要な関数のみを選択する方法も便利です。

「ヒートマップモード」では、データ確認時に関数名の表示が邪魔になることがあります。そこで「サンプルグラフ」リボンバーで「ショートネーム」または「関数名非表示」を選択できます。

14.1.8 セルフサンプル / サンプル合計 / 混合サンプル

このドロップダウンメニューはコールスタックデータを含む「コールスタック」プロファイルでのみ表示されます。このドロップダウンでサンプルのグラフ表示方法として、「セルフサンプル」、「サンプル合計」、または「混合サンプル」のいずれかを選択します。

- 「セルフサンプル」としてグラフ化される関数は、その関数がリーフ関数としてサンプルされるたびにサンプルマーカーを表示します(プログラムが停止されサンプリングされたときに、その関数が実行されていたことを示します)。
- 「サンプル合計」としてグラフ化される関数は、その関数がリーフ関数であるかどうかに関わらず、コールスタックの一部としてサンプルされるたびにサンプルマーカーを表示します。
- 「混合サンプル」としてグラフ化される関数は「サンプル合計」と同様のサンプルを表示しますが、リーフ関数のサンプル(「セルフサンプル」)は濃色で表示されます。

例えば `nnMain` 関数は、「セルフサンプル」でグラフ表示されたサンプルマーカーはほとんどありませんが、「サンプル合計」 または「混合サンプル」を選択すると多数のサンプルマーカーがグラフ表示されることがあります。

14.1.9 名前の表示幅の調整

関数またはスレッドをグラフ化すると、その名前が「サンプルグラフ」の左側に表示されます。「サンプルグラフ」ツールバーの「名前」ドロップダウンには、関数名やスレッド名の表示方法を操作する 4 種類の設定があります。

- 関数名非表示:関数名やスレッド名を描画せずにすべて非表示にします。
- ショートネーム:短縮した関数名を表示します(幅 100 ピクセルに制限)。スレッド名に関しては、スレッド ID のみを表示します。
- ロングネーム:引数なしの関数名を表示します。スレッド名は省略せずに表示します(ID と名前の両方)。
- フルネーム:引数を含む関数名を省略せずに表示します。スレッド名は省略せずに表示します(ID と名前の両方)。

14.1.10 複合グラフ

「複合グラフ」では、同じ横方向の列に描画された特定のコアに対するすべての関数およびスレッドを確認することができます。これにより、未選択の関数やスレッドの結果、発生したシーケンスやギャップを把握する際に役立ちます。

「複合グラフ」ドロップダウンボックスでは、組み合わせた関数およびスレッドグラフの描画動作を選択することができます。ドロップダウンメニューには 3 つのオプションがありますが、「複合グラフ非表示」がデフォルト設定です。

- 複合グラフ非表示:複合グラフは描画されません。
- 複合グラフ表示:関数やスレッドに対して複合グラフが描画されます。
- 複合グラフのみ表示:関数やスレッドに対して複合グラフが描画される一方で、関数やスレッドの各グラフは描画されません。

14.1.11 グラフツールチップ

「グラフツールチップ」ドロップダウンは関数の列、スレッド、または「アイシクルグラフ」関数にマウスオーバーしたときに確認したいツールチップの対応を指定できます。

- ツールチップ非表示:ツールチップを表示しません。
- 名前のみ表示:名前のみ表示します。
- 短い説明を表示:名前、サンプル統計、およびフレーム番号を表示します。
- 詳細説明を表示:名前、スレッド、システム分類、スパイク検出、周期性、サンプル統計、フレーム番号、プロファイル時間を表示します。
- **Fn** 時間表示:関数やコールスタックをポイントしている正確な時間で表示します。関数やスレッド列は無視するかわりに、プロファイル期間中のマウスの並列ポジションに焦点を当てます。

14.1.12 ラインハイライト

「ラインハイライト」ドロップダウンでは、「フレーム単位負荷」グラフに表示されるハイライトの種類を指定できます。

- ラインハイライト非表示:ハイライトは描画しません。
- 誤差:ハイライトはデータ間の誤差を示します。
- 最高・最低値:ハイライトはデータの最高値、最低値を示す横方向のバンドです。

14.1.13 サンプルギャップ

「サンプルギャップ」ドロップダウンではプロファイルのサンプルギャップを表示したり非表示にすることが可能です。サンプルギャップの特定は、プロファイラによるサンプル記録時間、そしてサンプル間のギャップが予測値より大幅に超過している箇所を確認することで行います。ギャップ発生の考えられる原因として、割り込みが無効になっていることが挙げられます。

- サンプルギャップ非表示:サンプルギャップは描画されません。
- サンプルギャップ表示:サンプルギャップを描画します。

14.1.14 サンプル、負荷、およびカウンタのスケール

「サンプル」、「負荷」、「カウンタ」グラフの縦サイズまたは縦軸のスケールは、「サンプルグラフ」ツールバーの「バーチカルスケール」ドロップダウンで設定します。ここで設定した変更内容は、次回プロファイラ起動時に反映されます。各「バーチカルスケール」ドロップダウンでは、折れ線グラフをすべて「非表示」にするオプションも選択できます。

グラフを表示、非表示にすばやく切り替える場合は、それぞれのラベル名(「サンプル」、「負荷」、または「カウンタ」)をクリックします。ラベル名を再度クリックすると、スケールが前回の値に戻ります。

実用的な縦軸の範囲は「1x」から「5x」です。y 軸にはより高い解像度とより多くの単位ラベルが表示されるため、特定の数値を正確に測定するためにはより大きな縦軸の目盛りを使用してください。

14.1.15 表示の操作

「サンプルグラフ」ツールバーには表示を変更するボタンがあります。

- 左向き矢印:画面を一番左端までスクロールします。
- 左右矢印:画面内にプロファイル全体が表示されるように横方向に縮小します。
- 右向き矢印:画面を一番右端までスクロールします。

14.2 タイムライン

「タイムライン」は「サンプルグラフ」の下にあります。フレームレートグラフとしてプロファイル全体をズームアウトして表示します。「タイムライン」の上のオーバーレイは、上記グラフの現在の拡大エリアを表すウィンドウになります。タイムライン上のウィンドウはスクロールバーと同様にクリック&ドラッグすることができます。またウィンドウの左右どちらかの側を一度クリックすると、ウィンドウ 1 画面の幅分だけ、その方向に移動します。マウスのホイールを使用して「タイムライン」画面をズームイン、ズームアウトすることもできます。

フレームレートグラフが「タイムライン」に表示されない場合は、ゲームがフレームに対して適切にマークされていないことを意味します。この方法の詳細は、ニンテンドー 3DS におけるフレームマーキングを参照してください。別の方法として、「ハートビート」タブで特定コアの特定ハートビートに対して、「フレームレートとして使用」を選択することで、「タイムライン」のフレームレートとしてハートビートを描画することも可能です。

14.3 ズーム調整

ズーム(横方向)はマウスホイールで操作できます。現在のズームレベルは「サンプルグラフ」ツールバーに表示されています。一番簡単に最大ズームアウトするには、「サンプルグラフ」ツールバーの「全体表示」ボタンをクリックします。

ズーム値は上下矢印キーでも調節できます。グラフをすばやくズームする場合に便利な方法です。

14.4 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

このメニューが表示されると、現在選択されている関数(オレンジ色の横長方形)、および現在の時間(オレンジ色の縦列)がハイライトされます。コンテキストメニューにおける特定の選択は現在の関数または現時間に影響するため、これらは参考のためにマークしています。

14.4.1 直上の親関数を展開

「直上の親関数を展開」オプションは現在選択中の関数の親関数を選択して、その親関数を現在選択中の関数の上に配置します。1 つの関数はさまざまな異なる親関数から呼ばれる可能性があるので、ここではすべての親関数が選択されます。

14.4.2 直下の子関数を展開

「直下の子関数を展開」オプションは現在選択中の関数の子関数を選択して、その子関数を現在選択中の関数の上に配置します。「サンプルグラフ」上の関数のいずれかをダブルクリックするだけでこのオプションを手早く設定できます。

14.4.3 直下の子関数の選択解除

「直下の子関数の選択解除」オプションは「直下の子関数を展開」オプションの動作をもとに戻します。

14.4.4 再配列

以下の「再配列」オプションがあります:

- 時間順:選択中のすべての関数を右クリックされたフレーム内で、発生した順番に基づいて再配列します。「ストーリー分析」で使用される配列方法と同じです。

- 量:選択中のすべての関数をサンプリング頻度に基づいて再配列します。「関数」タブ内で「合計」によって並び替えられた順序と同様です。
- 周期性:選択中のすべての関数をサンプリング周期的に基づいて再配列します。関数のサンプリングが周期的であるほど上部に表示され、サンプリングが不定期であるほど下部に配置されます。
- 関数名:選択中のすべての関数を関数名のアルファベット順に再配列します。あらかじめ関数名がわかっていて、「サンプルグラフ」内に多数の関数がある場合、特定の関数を見つける際に便利です。

14.4.5 選択解除

以下の「選択解除」オプションがあります:

- 現在の関数:現在ハイライトされている関数の選択を解除します。
- その他の関数すべて:現在ハイライトされている関数以外の関数の選択を解除します。
- 特定コアの関数すべて:現在ハイライトされている関数があるコアのそれ以外の関数の選択を解除します。
- その他のコアの関数すべての選択解除:現在ハイライトされている関数があるコア以外の関数すべての選択を解除します。
- 現在の関数以上:現在ハイライトされている関数以上にあるすべての関数の選択を解除します。
- 現在の関数以下:現在ハイライトされている関数以下にあるすべての関数の選択を解除します。

14.4.6 移動

以下の「移動」オプションがあります。

- 上へ移動:現在ハイライトされている関数を 1 つ上の列に移動します。
- 下へ移動:現在ハイライトされている関数を 1 つ下の列に移動します。

14.4.7 アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブ に焦点を移動して、クリックした関数のアセンブリを表示します。

14.4.8 詳細表示

「詳細表示」オプションは新規の関数詳細ウィンドウを開いて、クリックした関数の詳細を表示します。

14.4.9 該当時間の該当コア

以下の「該当時間の該当コア」オプションがあります。

- サンプリングされたリーフ関数を検索:現在ハイライトされている時間における該当コアのサンプリングされたリーフ関数を選択します。関数は現在表示中の関数の 1 番上に配置されます。
- サンプリングされた関数すべてを検索:現在ハイライトされている時間における該当コアのすべてのサンプルされた関数(コールスタック)を選択します。これらの関数は現在表示中の関数の 1 番上に配置されます。

14.4.10 該当時間のすべてのコア

以下の「該当時間のすべてのコア」オプションがあります。

- サンプリングされたリーフ関数を検索:現在ハイライトされている時間におけるすべてのコアのサンプリングされたリーフ関数を選択します。これらの関数は現在表示中の関数の 1 番上に配置されます。
- サンプリングされた関数すべてを検索:現在ハイライトされている時間におけるすべてのコアのサンプルされた関数(コールスタック)を選択します。これらの関数は現在表示中の関数の 1 番上に配置されます。

14.4.11 該当ハートビートのみ選択

「該当ハートビートのみ選択」オプションはすべての関数および時間の選択を解除してから、マウスオーバーしている現在のフレームを選択します。「ストーリー分析」または「スパイク検出」で特定のフレームにすばやく焦点を当てたいときに便利なオプションです（まずフレームを選択してから分析を有効にします）。

もし 1 つ以上のフレームを選択したいときは、「サンプルグラフ」ツールバーでマウスモードを「フレーム選択」に変更してから、左マウスボタンでフレームの選択または選択解除をします。

15 コードカバレッジタブ

「コードカバレッジ」タブは、特定のプロファイル内でサンプリングされた関数とサンプリングされていない関数の追跡が可能です。この機能はテスト範囲を決定するとき、または予期しない不測の動作を検出するときに有効です。

保存済みプロファイルを読み込むと、このリストは消去されその保存済みプロファイルに基づいてリストを再度算出される点に注意してください。このリストはプロファイルと一緒には保存されませんが、その代わりプロファイルに基づいて生成されます。

15.1 プロファイル中検出された関数

このウィンドウにはプロファイリング中に検出された関数の名前が表示されます。リスト中の関数の数は、コード合計サイズと一緒にヘッダにレポートされます。コード合計サイズはフィルターが未設定の場合にのみ表示されます。

15.1.1 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

15.2 プロファイル中検出されなかった関数

このウィンドウにはプロファイリング中に検出されなかった関数名が表示されます。リスト中の関数の数は、コード合計サイズと一緒にヘッダにレポートされます。コード合計サイズはフィルターが未設定の場合にのみ表示されます。

15.2.1 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

15.3 関数のフィルタリング

テキストボックスに何らかの文字列を入力すると、その文字列を含むアイテムを大文字小文字を同一視して検索します。検索文字列に一致するものだけが表示されます。フィルタリング結果を反転する（そのフィルタとマッチしないものすべてを表示する）には、テキストボックスの右側の感嘆符ボタンをクリックします。一致するアイテムがない場合、テキストボックスがオレンジ色にハイライトされます。

また正規表現にも対応しています(.NET RegEx 利用、Perl 5 互換)。フィルターで文字を正規表現として解釈するためには、テキストボックス右側の「RegEx」ボタンをクリックしてください。また基本的なフィルターとは異なり、正規表現では大文字・小文字の区別を行うことに注意してください。

便利な関数フィルターは便利な簡易フィルターおよび便利な正規表現フィルター表を確認してください。正規表現の詳細については正規表現クイックガイドを参照してください。

15.3.1 保存済みフィルター

保存フィルターはプロファイルセッション間で同じフィルターを使用可能にします。プロファイルが開いている時に保存フィルターを読み込みます。デフォルトでは、「フィルタ」ドロップダウンボックスでカスタマイズ可能なフィルター例が表示されます。ドロップダウンボックスに新たなフィルタを追加する場合は、フィルタ文字列を入力してから「保存」(「+」)ボタンを押します(「反転」と「RegEx」ボタンの状態もフィルタ文字列と共に保存されます)。ドロップダウンボックスからフィルタを削除する場合は、まずそのフィルタを選択してから「削除」(「-」)ボタンを押します。

> 補足:「関数」タブで保存したフィルターは「コールツリー」タブと「コードカバレッジ」タブからも利用できます。ただし「スレッド」タブと「カウンタ」タブの「フィルタ」ロップダウンボックスは共有されないことに注意してください。これはスレッド名またはカウンタ名を検索するタブであり、関数の検索には使用されないためです。

15.4 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

15.4.1 関数名をクリップボードにコピー

引数付きの関数フルネームをクリップボードにコピーします。

15.4.2 関数の選択

「関数の選択」オプションは、クリックした関数を「関数」タブ上で選択状態にします。

15.4.3 アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブに焦点を移動して、クリックした関数のアセンブリを表示します。

15.4.4 詳細表示

「詳細表示」オプションは新規の関数詳細ウィンドウを開いて、クリックした関数の詳細を表示します。

16 アセンブリタブ

「アセンブリ」タブは、関数のどの部分をサンプルしたかを確認することが可能です。この機能は関数内で発生した特定の問題箇所を見つけ出すのに役立ちます。

16.1 アセンブリタブのツールバー

「アセンブリ」タブのツールバー内の操作は以下のとおりです。

16.1.1 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

16.1.2 表示の操作

サンプル数がわずかしかない、大きめの関数をより早く調べるためには、「戻る」および「次へ」ボタンを利用することができます。このボタンを使用すると、セルルまたはサブのいずれかにおいてゼロでない数値を保持するアセンブリの前の行または画面外の次の行にスクロールすることができます。

16.1.3 ソース表示

アセンブリと交互にソースコードを表示することが可能です。必要な情報はハードドライブに保存されている ELF ファイルから取得します。このためソースコードの表示に関連していくつか制限事項があります。

このボタンはオン／オフ式です。プロファイラのスタートアップ時のデフォルトはオフです。ボタンがオンの場合、新規に表示する関数を選択すると、その関数のソースコードを自動的に取得しようとします。

> 補足：「ソース表示」をクリックすると、「アドレス」に対する並び替えが実行されてから、ソースコードをインターリーブします。

16.1.3.1 インターリーブソースコード表示における制限事項

インターリーブソースコード表示におけるいくつかの制限事項をここに挙げます。

- コンピューターに Cygwin ツールパッケージの `binutils` がインストールされている必要があります。プロファイラはパッケージ内の `addr2line` ツールを利用して、各アセンブリ命令に関連するソースコードの行を取得します。`binutils` パッケージが見つからない場合は、「`Cygwin インストレーションで addr2line が見つかりません。`」`binutils パッケージをインストールしてください。`」というメッセージが表示されます。
- プロファイル初回取得時と同じコンピューターの場所に、オリジナルの ELF ファイルを保存しておく必要があります。必要な ELF ファイルにアクセスできないときは「（*`ElfName`*`が見つかりません。`」のメッセージが表示されます。

- PC 上の ELF ファイルの日付はプロファイル取得日時よりも古くなければなりません。プロファイル取得日時よりも ELF ファイルの日付が新しい場合は、ELF ファイルと現在のプロファイルデータの関連付けができません。この場合は、「ELF ファイルの修正日時がプロファイルの日時よりも新しくなっています。」のメッセージが表示されます。
- 個々のソースファイルのタイムスタンプは確認されません。そのため、ELF ファイル作成後にソースファイルが修正されたり、行番号が変更されていたりする可能性があります。この場合はアセンブリ表示内で間違った行が表示されてしまいます。インターリーブソースコード表示を使用する場合は、この制限に注意してください。

16.2 表示する関数の選択

「アセンブリ」タブを開き、「関数」タブにある関数をクリックすると、その関数の情報が「アセンブリ」タブに表示されます。「アセンブリ」タブ　ツールバーの「選択された関数」テキスト右側に現在選択されている関数が表示されます。ただし選択された関数は「関数」タブ上ではハイライトされません。

RPL 内部関数は表示できません。もしこれらの関数の 1 つが選択された場合は、「アセンブリ」ウィンドウ内にその関数の「セルフ」および「サブ」の計測カウントとともにメッセージが表示されます。これらのカウントは「関数」タブに表示されるカウントと同じ数値になるはずです。

16.3 アセンブリ表示

アセンブリのソースコードがこのタブ領域に実際に表示されます。ソースコードの各行に記載される情報は以下の通りです。

- アドレス:メモリ内でアセンブリの行を表すアドレスです。
- セルフ:プログラムカウンタがそのアドレスでそのアセンブリの行を実行しているときに取得したサンプルの数です。
- サブ:「コールツリー」内のサンプルされた命令よりも上位にこの関数がある場合に取得したサンプルの数です。このためこの数値は「コールスタック」プロファイルのリンクつき分岐命令上においてのみ表示されます。
- アセンブリ:そのアドレスに格納されているアセンブリ命令です。

ニンテンドー 3DS CPU プロファイラでは、コンパイラツール提供の逆アセンブラとは異なる逆アセンブラを使用しています。このためコンパイラツールのバージョンとアセンブリの直接比較を行うと、多少異なるテキストでの結果となりますが、これは使用されるニーモニック内の差分と考えるべきです（または使用されていないケースも考えられます）。

列はサイズ変更や並び替えが自由にできますが、ただし「コピー」機能を使用する場合は、デフォルトの列順とサイズが使用されます。

16.3.1 ハイライト機能

プロファイル中にアセンブリのどの箇所がプロファイルされているかを視覚的に評価しやすくするために、「セルフ」のサンプルが最も多い行を濃い赤で表示しています。「セルフ」のヒット回数が低下するにつれて赤色が薄くなります。「セルフ」サンプルよりも「サブ」サンプルが多い箇所は、黄緑の濃淡の表示となります。

16.3.2 ソート機能

基本的なソート（並び替え）機能がアセンブリ表示で利用可能です。デフォルトのソート設定は「アドレス」による昇順です。表示対象の新規関数が選択されると、ソートモードは必ずデフォルト設定に戻ります。

ソートを変更するには、ソート対象の列のヘッダーをクリックします。各列は 1 方向にのみ並び替え可能です。「アドレス」および「アセンブリ」では昇順で並び替えを行い、「セルフ」および「サブ」の両方では、降順で並び替えを行います。

補足： 列ヘッダのいずれかをクリックすると、「ソース表示」は自動的に無効になります。

16.4 右クリックコンテキストメニュー

タブ内で右クリックするとコンテキストメニューが表示され、これには様々なオプションが含まれます。

アセンブリ命令が分岐の場合のみ特定の関数を有効にするオプションです。

関数内にリテラルプールがある場合にのみデータを有効にするオプションです。

16.4.1 対象関数の選択

「対象関数の選択」オプションは、クリックした関数を「関数」タブ上で選択状態にします。

このオプションは対象関数がプロファイル中に確認されている場合にのみ有効です。対象関数が未確認の場合は、オプションは無効になります。

16.4.2 対象へジャンプ

「対象へジャンプ」オプションを選択すると、「アセンブリ」表示で対象アドレスを表示します。ジャンプ操作が同じ関数内で発生すると、「アセンブリ」表示では対象アドレスがウィンドウ最上部にスクロール移動します。別の関数へジャンプした場合は、「アセンブリ」表示は選択関数を対象関数として設定します。

16.4.3 対象詳細の表示

「対象詳細の表示」オプションは新規の関数詳細ウィンドウを開いて、対象関数の詳細を表示します。

16.4.4 対象関数名のコピー

「対象関数名のコピー」オプションは引数付きの関数フルネームをクリップボードにコピーします。

16.4.5 データタイプ別表示

関数にリテラルプールが含まれている場合、プロファイラはリテラルプール内のデータを確認して、そのデータタイプを自動的に特定します。「データタイプ別表示」オプションでは、自動割り当てタイプをオーバーライドできます。割り当て可能なタイプは以下になります。

- **Int32**：符号付き 32 ビット整数
- **UInt32**：符号なし 32 ビット整数
- **Single**：32 ビット浮動小数点
- **String**：ASCII (ISO 8859-1) 拡張文字の 4 バイト配列
- 関数：データを関数アドレスとして取り扱う

17 コンソールタブ

「コンソール」タブでは、開発キットからのデバッグ出力を表示することができます。出力を表示するためには、開発キットとの接続を「有効」ボタンでオンにしておく必要があります。

この「コンソール」タブはコンソールのウィンドウのように動作します。新しい出力はウィンドウの下の部分に表示されます。およそ 1000 行をバッファリングでき、これを超えると古い出力から消去されてゆきます。

17.1 コンソールタブのツールバー

「コンソール」タブのツールバー内の操作は以下のとおりです。

17.1.1 有効ボタン

デフォルトでは、「コンソール」で開発キットからのデバッグ出力は表示しません。デバッグ出力を表示するには「有効」ボタンを押してオンにする必要があります。デバッグ出力を停止するには、「有効」ボタンを再度押してください。

17.1.2 クリアボタン

「クリア」ボタンを押すと、現在のウィンドウ内のすべてをクリアします。

17.1.3 コピーボタン

「コピー」ボタンを使用して、現在のソート設定やフィルタリング設定を反映した表示内容をクリップボードにコピーすることができます。コピーした内容はメール、ドキュメント、Excel スプレッドシートなどへの貼り付けが可能です。コピー内容はタブ区切り形式を使用しているので、Excel スプレッドシート画面に貼り付ける場合でも列の構成を維持することができます。

「コピー」ボタンは現在の選択内容を反映しています。「コピー」ボタンを押した時点で選択されている項目があれば、その項目のみがコピーされ、もし何も選択されていない場合は、すべてがコピーされます。

17.1.4 テキストフォーマットの選択

「テキスト形式」ドロップダウンでは、開発キットから出力したテキストのデコードに使用するフォーマットを変更できます。このルールはテキスト受信時に適用されるもので、遡及的に適用されるものでありません。

有効な選択項目は以下になります。

- **UTF-8**
- **Shift-JIS**

18　関数詳細ウィンドウ

「関数詳細」ウィンドウには選択関数に関する追加情報が表示されます。以下の情報が表示されます。

- 関数名
- 関数が取得されたモジュール
- 関数アドレス
- 関数サイズ(バイト)
- 関数内の命令数
- 対象関数が呼び出す関数
- 対象関数を呼び出す関数

また、関数がプロファイル中に検出された場合は、以下の詳細も表示されます。

- 対象関数の最小「合計」% (表示対象は「コールスタック」プロファイルのみ:すべてのスレッドおよびコアに対するコールツリー内の最大「合計」% から求められる数字)
- 対象関数の「セルフ」% (すべてのスレッドおよびコアにわたるすべての「セルフ」% の合計から求められる数字)
- 関数が検出されたスレッド

18.1　表示する関数の選択

「関数詳細」ウィンドウで表示対象の関数を選択するには、コンテキストメニューを右クリックします。詳細確認対象の関数を右クリックして、「詳細表示」を選択します。これで「関数詳細」ウィンドウが表示されます。プロファイラのその他の機能を継続して使用する場合は、このウィンドウは閉じる必要があります。

プロファイラの以下のタブにはコンテキストメニューオプションがあり、関数の詳細を表示できます。

- 関数タブ
- チェッカータブ (結果表では一部のみが対象で、すべてには提供されていません)
- チェッカータブ
- コールツリータブ (通常およびリバースコールツリー両方)
- サンプルグラフタブ
- コードカバレッジタブ

18.2　関数詳細ウィンドウの操作

「関数詳細」ウィンドウの操作は以下のとおりです。

18.2.1　アセンブリ表示

「アセンブリ表示」オプションは アセンブリタブに焦点を移動して、クリックした関数のアセンブリを表示します。

18.2.2　関数の選択

「関数の選択」オプションは、クリックした関数を「関数」タブ上で選択状態にします。

18.2.3 関数呼び出しのトラバース

「関数詳細」ウィンドウで表示対象の関数を変更するには、「呼び出し先関数」または「呼び出し元関数」リスト内で関数をダブルクリックします。この操作により、選択関数の詳細表示を切り替えます。本手順は繰り返し実行可能です。

19 システム分類

各関数は自動的に属するシステムに分類されます（物理、グラフィック、音声など）。この機能では主に、各システムの一般的に認められている用語と一致する文字列を使用して分類しています（例えば、物理では "broadphase"、演算では "matrix" など）。もちろん一部の関数が間違って分類されてしまうケースもありますが、たいていの場合は正しく分類されるはずため、分析の際に非常に便利です。

システム分類は主に以下の 3 つの機能で使用されます。

- 関数のハイライト：「カラーコーディング」が「システム」に設定されている場合、選択される関数すべてがシステム分類にしたがって色分けされます。
- アイシクルグラフ：「アイシクルグラフ」で表示される関数すべては、システム分類にしたがって色分けされます。
- スタックグラフ：「スタックグラフ」で表示される関数すべては、システム分類にしたがって色分けされます。
- クイックフィルター：「クイックフィルター」はシステム分類を使用して各関数クラスを特定します。

19.1 分類のカスタマイゼーション

関数をシステム毎に分類する方法は、`bin\QuickFilters\Classifications.xml` ファイル内の正規表現を使用します。デフォルト設定では一部コードを見落としてしまう場合もあるので、使用中の正規表現を置き換える方法を提供しています。この方法はクイックフィルターで使用される方法と極めて類似しています。

使用中の正規表現を変更する方法：

(1) 現在使用中の `Classifications.xml` ファイルのコピーを作成して、そのコピーファイルをカスタムボタンディレクトリがあるフォルダにおきます。

(2) コピーから編集対象の設定以外の分類設定を削除します。

(3) 分類ファイル内で正規表現を編集します。

現時点では、分類設定のみが編集可能です。しかし、将来的には新規の分類を設定できる機能を予定しています。

> 補足：カスタム分類設定はデフォルト分類より優先されます。同様に、分類ファイルの最初の定義が使用されます。

20 パフォーマンスカウンタグループ

CPU には、CPU 内部のイベントカウントを設定可能な 4 つのパフォーマンスカウンタがあります。複数のパフォーマンスカウンタがあるため、プロファイラでは「パフォーマンスカウンタグループ」を使用して、類似するカウンタや、補完的なパフォーマンスカウンタを分類しています。プロファイラではサンプルを取得するたびに、特定のパフォーマンスカウンタグループ(コアごと)を記録できます。このセクションでは各パフォーマンスカウンタグループの詳細と、それぞれが記録するデータの種類を説明します。

20.1 パフォーマンスカウンタ無効設定

デフォルトの設定になります。この設定のときは、パフォーマンスカウンタのデータはプロファイルデータには記録されません。

20.2 命令ミスおよびバス競合

対象コードがキャッシュをスラッシングしているかの判断に利用します。さらに命令ミスごとのサイクルが 32 以上の場合は、バスが(GFX、音声などで)非常にビジーであることを表します。以下の2 種類のグラフをこれらのカウンタから作成します。「命令キャッシュミス」と「命令ミスごとのサイクル」。

命令キャッシュミスの分析

命令キャッシュミスが増加すると、キャッシュ内でコードがスラッシングしている可能性が高くなります。しかしこのグラフによって確実に問題点が明らかになるというわけではありません。命令キャッシュミスはサンプリングとサンプリングの間に累算するため、グラフは取得サンプル数に極めて影響されやすいという点に注意してください(このグラフは、サンプリングレートに影響されやすい唯一のパフォーマンスカウンタグラフです)。このようにサンプリングレートに影響を受けやすいため、例えば命令キャッシュミス 5000 回が良い結果または悪い結果であるかどうかを、グラフ全体を確認して最小、最大、平均値と比較せずに判断することは困難です。基本的には、グラフの数値が高いとスラッシング頻度が高いことを示します(キャッシュに対しコードが大きすぎる可能性があります)。値が高い場合、コードのサイズを減らしてみてください。対策としては、Thumb code の使用、(スピードの最適化でなく)サイズの最適化、共通サブルーチンの再利用、およびインラインの無効化または低減などがあげられます。

命令ミスごとのサイクルの分析

このグラフは命令キャッシュラインが一杯になるのを待つ際の CPU サイクルの回数をカウントします。CTR では 1 つのキャッシュラインにつき最小 32 サイクルです。最初のグラフとは異なり、このグラフは正規化されており、サンプリングレートに関係なく一貫性のある数値になります(最小値となる 32 は非常に良い結果を表し、80 から 100 以上は悪い結果になります)。実際にはこのグラフはバスの状態を示します(バスの状態は直接計装できないため、このメトリックはバスを計装する代替策となります)。結果として、このグラフによってフレーム内の異なる時点でバスがどの程度ビジーかが把握できます。バスがビジーの場合は、GPU、DSP／オーディオ、ネットワーク、カード、LCD などの他のクライアントによるものです。例えば、ディスプレイバッファが FCRAM にある場合と、VRAM にある場合ではバスの状態にはっきりと違いが確認できます(FCRAM の場合にはバス待機時間がより長くなります)。

20.3 命令および I-キャッシュ効率

このメトリックでは命令キャッシュラインが一杯になるまでの待機時間の測定に役立ちます。さらに、全体的なシステム効率性を測定するサイクルごとの命令も表示します。以下の 2 種類のグラフをこれらのカウンタから作成します。「命令待ちでストールしたサイクル数比率」および「サイクルあたり命令数」。

命令待ちでストールしたサイクル数比率の分析

CTR では一般的に CPU サイクルの約 30% は、キャッシュに読み込まれる命令待ちに浪費されます（30%以下の場合、memcpy のようにコードがタイトなループ内にあることを示します）。他のシステムからのトラフィックが増加すると、CPU がメインメモリから命令を読み込むまでの時間が長くなります。本グラフは正規化されるので、サンプリングレートにかかわらず数値は一貫しています。「命令ミスごとのサイクル数」グラフは各命令がキャッシュに読み込まれるまでの実際の待機時間を表示するのに対し、本グラフは待機時間と CPU が利用可能な時間の相関関係の理解に役立ちます。この関係は CPU + キャッシュ + バス + FCRAM の効率性や、相互に調和して動作しているかの判断基準になります。

サイクルあたり命令数の分析

このグラフは全体的な CPU 効率の判断基準になります。CTR の場合 1.0 以上が理想ですが通常値は 0.2 程度になります（これらは現在までのゲームに基づいた値で、これまでの計装最大値は 0.55 です）。サイクルあたり命令数値が低くなる要因は数多くあります。例えばメモリ待機、前の計算完了待機、コプロセッサ待機（例えば VFP など）などがあげられます。そのため、このグラフでフレームの問題箇所を検出し、他のグラフで原因を調査するようにしてください。

20.4 メモリおよび演算パフォーマンス

このメトリックは命令やデータ待ちによる CPU ストール時間を測定します。メモリからのデータ読み込み時、または演算結果待ちの間にデータストールが発生します。以下の 3 種類のグラフをこれらのカウンタから作成します。「命令キャッシュミスによるストール率」、「データハザードによるストール率」、および「未検出のデータ/命令によるストール率（下限）」。最後のグラフは最初の 2 つのグラフの最大値になります。（現在の CPU パフォーマンスカウンタではストールの原因となるすべての問題に対する正確な比率を算出することはできません。下限のみ算出できます。）

命令待ちでストールしたサイクル数比率の分析

CTR では一般的に CPU サイクルの約 30% は、キャッシュに読み込まれる命令待ちに浪費されます（30%以下の場合、memcpy のようにコードがタイトなループ内にあることを示します）。他のシステムからのトラフィックが増加すると、CPU がメインメモリから命令を読み込むまでの時間が長くなります。本グラフは正規化されるので、サンプリングレートにかかわらず数値は一貫しています。「命令ミスごとのサイクル数」グラフは各命令がキャッシュに読み込まれるまでの実際の待機時間を表示するのに対し、本グラフは待機時間と CPU が利用可能な時間の相関関係の理解に役立ちます。この関係は CPU + キャッシュ + バス + FCRAM の効率性や、相互に調和して動作しているかの判断基準になります。

データ読み込み待機によるストール率割合の分析

このグラフはデータハザードにより CPU 内のサイクルストール率を表示します（メインメモリからのデータ待機中や演算結果待機中など）。グラフ中で急激なスパイクとなる部分は、メモリ内のデータ整理、データサイズの縮小、演算中やデータ取得中に別作業を同時処理するなどで改善の余地のある部分です。

未検出のデータ/命令によるストール率（下限）の分析

このグラフは最初の 2 種類のグラフ上の最大値を示します。命令キャッシュミスやデータハザードによって CPU がストールした比率の下限を表します。命令キャッシュミスやデータハザードは同時に発生することもある

ため、上記 2 種類のグラフを加算するのは誤りです。代わりに上記 2 種類のグラフ から最大値を抜き出すことにより、CPU が何らかの理由で待機していた最小値を表示できます（しかしグラフで表示される数値よりも長く待機している可能性が大きいです）。

20.5 データキャッシュ読み込みパフォーマンス

このグラフはデータキャッシュ内の読み込みミスによりメインメモリからの読み込みが発生している比率の測定に役立ちます。以下の1 種類のグラフをこれらのカウンタから作成します。「データキャッシュ読み込みミス率」。

データキャッシュ読み込みミス率の分析

このグラフはデータキャッシュ内の読み込みミスのためにメインメモリから読み込みとなる比率の判断材料になります。本来はこの発生率は非常に低い値であるべきです。発生率にスパイクがある場合、処理中のデータセットが大きすぎてキャッシュ内に収まらない、データセットの空間的位置が悪い（メモリ内でやりとりしている）、またはデータセットがデータ内のシングルパスに対して読み込まれている（再利用されていない）ことなどが考えられます。

20.6 データキャッシュ書き込みパフォーマンス

このグラフはデータキャッシュ内の書き込みミスによるメインメモリからの読み込み発生率の測定に利用します。以下の 1 種類のグラフをこれらのカウンタから作成します。「データキャッシュ書き込みミス率」

データキャッシュ書き込みミス率の分析

このグラフはデータキャッシュ内の書き込みミスによるメインメモリからの読み込み発生率の測定に役立ちます（キャッシュラインを変更可能にする前に必ずキャッシュラインを読み込んでおく必要があります）。本来はこの発生率は非常に低い値であるべきです。発生率にスパイクがある場合、データセットの空間的位置が悪い（メモリ内でやりとりしている）、またはデータセットがデータ内のシングルパスに対して書き込まれている（再書き込みされていない）ことなどが考えられます。

20.7 ロード／ストアキューのプレッシャー

このメトリックはコードの特定部分でロード／ストア処理の数が多すぎかどうか（キューが一杯かどうか）の測定に利用します。以下の 1 種類のグラフをこれらのカウンタから作成します。「ロード／ストアキューが一杯でストールする率」。

ロード／ストアキューが一杯でストールする率の分析

このメトリックはコードの特定部分でロード／ストア処理の数が多すぎかどうか（キューが一杯かどうか）の測定に利用します。コードが一度に多数のロード／ストアを試みているため、間隔を設けることで少しでもこのタイプのストールを回避するべきであることを示しています。具体的に、この機能は関数やメモリトラフィックで混み合っているコード部分の特定に役立ちます。

20.8 分岐予測ミス

このメトリックは過度の分岐予測ミスが発生しているかどうかの測定に利用します。この結果によっては特定のロジックテストを反転することでパフォーマンスが向上する可能性があります。以下の 1 種類のグラフをこれらのカウンタから作成します。「分岐予測ミス発生率」。

分岐予測ミス発生率の分析

このメトリックは過度の分岐予測ミスが発生しているかどうかの測定に利用します。この結果によっては特定のロジックテストを反転することでパフォーマンスが向上する可能性があります。しかし分岐予測ミスの発生率はグラフの表示値より大幅に高くなりますが（12% が実際は 80% という場合があります）、グラフの相対形は正確です。すべての分岐は条件的分岐であるかないか（分岐予測ミス／すべての分岐）にかかわらず測定されるため、誤予測された分岐のスケールは見掛けでは低くなります。「分岐予測ミス／すべての条件的分岐」を作成するパフォーマンスカウンタや複数パフォーマンスカウンタの組み合わせが存在しないため、正確なグラフは作成できません。分岐予測ミスを計算・表示する方法としてはこのグラフが最適です。

20.9 サンプリングされた関数とパフォーマンスカウンタの相関関係

サンプリングされた関数とパフォーマンスカウンタデータの間には非常に弱い相関関係しかありません。そのため一致するデータをあまり重要視しないように注意が必要です。例えば、strcmp 関数のサンプリングと同じタイミングで、データキャッシュ書き込みミスの比率が大きかったとします。この場合、strcmp 関数が書き込みミスを引き起こした可能性は低いと言えます。パフォーマンスカウンタは、サンプリングとサンプリングの間のデータを蓄積しますが、その間には多くの関数が動作しています。このことからちょうどサンプリング時に、プログラムカウンタがたまたま strcmp 関数内にあり、パフォーマンスカウンタが記録、クリアされたことが原因と思われます。サンプリングレートが高いほど、相関関係が強くなります。しかしプロファイラ記録コードが結果に著しい影響を与えます（他のメトリックスに対する干渉だけでなく余計な命令ミス、データキャッシュミスなどが発生します）。

パフォーマンスカウンタに問題を起こす関数を特定できなくても、パフォーマンスカウンタのデータがどのシステムや領域、コードに起因するかを判断することは可能です。例えば、レンダリングする直前にデータキャッシュ読み込みミスが大きな比率となった場合、自身のゲームの知識や呼び出された関数を手がかりに、そのミスはレンダリングする前に実行されたコードのためと判断できるでしょう。例えば、特定のシステムがレンダリング前に実行されていることから、データキャッシュ読み込みミスの原因はアニメーションシステムだと判断できます。

「ヒートマップ」モードを利用すると、サンプリングされた関数とパフォーマンスカウンタの相関関係が大幅に強調されるため、信頼性が若干向上する場合もあります。

21 コマンドラインオプション

プロファイラ GUI の機能の一部を操作するために、コマンドラインオプションを用意しています。本章ではコマンドラインのフォーマットについて説明します。

21.1 コマンドラインフォーマット

```
"Nintendo CPU Profiler.exe" [--options] [NPROF] [ELF] [Executable]
```

[--options]

選択可能なオプションで説明するオプション（指定なしまたは複数選択可）

[NPROF]

前回保存した NPROF（`C:\path\file.nprof`）のファイル名を入力してプロファイラ起動時に自動的にロードされるようにします。

[ELF]

接続処理に使用する AXF ファイルへのパスを入力します (C:\path\example.axf)。--loadelf オプションと同機能になります。「接続」オプションが利用可能になるとすぐに、指定ファイルを接続するように GUI に伝えるコマンドです。Executable が指定されていない場合は、プロファイラは自動検索します。

[Executable]

接続処理に使用する CCI または CCL ファイルへのパスを入力します (C:\path\example.rpx)。「接続」オプションが利用可能になるとすぐに、指定ファイルを「接続」するように GUI に伝えるコマンドです。ELF が指定されていない場合は、プロファイラは自動検索を試みます。

> 重要：NPROF パラメータには ELF または Executable パラメータを指定しないでください。

21.2 選択可能なオプション

--autosave[=filename]

ボタンを押す代わりに、プロファイル終了後に結果を直ちに保存するよう GUI に伝えるコマンドです。このオプションで保存するのは NPROF のみです。

コマンドラインのファイル名の指定はオプションです。指定されていない場合、現在読み込まれている ELF に基づいたファイル名を ELF と同じディレクトリに保存します。最終的なファイル名は以下のようになります。`filename-YYYYMMDD-hhmmss.nprof`

--loadelf=filename

（廃止）このオプションはコマンドラインでファイルを直接指定する方法に変更されました。「接続」オプションが利用可能になるとすぐに、指定ファイルを接続するように GUI に伝えるコマンドです。

--saveonerror

（廃止）このオプションを指定するとデータ解析に問題が発生していたとしても NPROF を保存します。このオプションの主な目的は、有効と思われるデータの解析に問題があった場合にプロファイラチームへデータの送信を可能にすることです。（本機能は便利すぎるため、オプション設定のままにしておく必要がないと判断されました。今回から常に有効な設定となっています。）

--script=filename

プロファイラ起動時に指定のスクリプトを自動的に実行します。この操作は指定の NPROF が読み込まれた後に実行されますが、-loadelf オプションが完了する前に発生することもあります。

--shutdown:

プロファイル取得後プロファイラ GUI を終了するオプションです。このオプションは -autosave オプション指定時のみの使用を推奨します。

22 スクリプト記述

プロファイラは GUI の自動操作を実行するために Windows PowerShell スクリプトファイルの読み込みに対応しています。これは同じプロファイルセットを複数回取得する場合に便利です。

22.1 Windows PowerShell

Windows PowerShell スクリプト機能ではかなり複雑な挙動が実現可能です。ただしこのスクリプト言語についての説明は、本ドキュメントのスコープ外となります。PowerShell スクリプト言語についての詳細情報は、Microsoft から入手可能です。

22.2 コマンドレット

プロファイラはスクリプティングで新規のコマンドレット(cmdlet)を作成して使用することが可能になります。すべての内蔵コマンドレットも利用可能です。別途にインストールされた拡張コマンドレットは利用できない場合があります。

すべてのプロファイラコマンドレットには「PROFILER-」の接頭辞が付きます。

> 補足： Windows PowerShell では、コマンドレットは大文字・小文字を区別しません。

プロファイラコマンドレットの引数はスペースで区切られ、引用符で囲まれていないテキストは内部比較用に大文字に変換されます。このため関数名との間で問題が発生してしまうので、すべての関数名を二重引用符(“”)で囲む必要があります。二重引用符(“”)を使用すると、複数の引数およびテンプレート情報を含む C ++ 関数名において必要に応じてスペースを入れることが可能になります。

もし引数名が [] で囲まれている場合は、引数名に渡すのは任意です。引数名が省略されている場合は、引数を渡す順番は表示順どおりでなければなりません。

22.2.1 PROFILER-ANALYZE

プロファイルデータの分析をリクエストします。

ストーリー分析

ストーリー分析で説明するストーリー分析モードを使用してデータ分析を行います。

```
PROFILER-ANALYZE -Story <StoryType>
```

-Story <StoryType>

ストーリー分析オプションのいずれか。オプションは ストーリータイプで説明しています。

スパイク検出

スパイク検出で説明するスパイク検出モードを使用してデータ分析を行います。

```
PROFILER-ANALYZE -SpikeDetection <SpikeDetectionType>
```

-SpikeDetection <SpikeDetectionType>

スパイク検出オプションのいずれか。オプションは スパイク検出タイプで説明しています。

例

```
] PROFILER-ANALYZE -Story Full
PROFILER-ANALYZE -SpikeDetection Rare
```

22.2.2 PROFILER-ASSEMBLY

「アセンブリ」タブの内容を設定するオプションを操作します。

設定関数

```
PROFILER-ASMFUNCTION -FunctionName <String>
```

-FunctionName <String>

使用する文字列がプロファイラに表示される関数名と正確に一致していなければなりません。C++ を使用している場合は、引数およびテンプレート情報もここに含まれます。

ソース表示

```
PROFILER-ASSEMBLY -ShowSource <Boolean>
```

-ShowSource <Boolean>

ソースコードを表示するかどうか指定します。

例

```
PROFILER-ASSEMBLY -FunctionName "TestFunction"
PROFILER-ASSEMBLY -ShowSource $true
```

22.2.3 PROFILER-CALLSTACKS

サンプルプロファイルにコールスタックを記録するかどうかを設定します。

```
PROFILER-CALLSTACKS [-ShowCallstacks] <Boolean>
```

[-ShowCallstacks] <Boolean>

コールスタックを記録するかどうか指定します。

例

```
PROFILER-CALLSTACKS $false
```

22.2.4 PROFILER-COPY

指定したタブの内容をクリップボードにコピーするようにプロファイラに指示します。

```
PROFILER-COPY [-Tab] <ScriptingTabs> [-ExtendedParam
<ScriptingCopyExtensions>]
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

[-ExtendedParam <ScriptingCopyExtensions>]

任意です。複数のコピーオプションがあるタブもあります。拡張オプションの一覧については、コピー拡張を参照してください。

例

```
PROFILER-COPY Functions
PROFILER-COPY CallTree Selected
PROFILER-COPY CodeCoverage -ExtendedParam Seen
```

22.2.5 PROFILER-FILTER

指定タブにフィルターを設定します。

このコマンドレットには少数の異なる形式があります。すべてのタブと互換性がない形式もあります。

By Name（名前によるフィルター）

PROFILER-FILTER のデフォルトフォームです。修正対象のタブ上の Filter/Find ボックスの結果になります。

```
PROFILER-FILTER [-Tab] <ScriptingTabs> [-Negate] <Boolean> [-RegEx]
<Boolean> [-FilterName] <String>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

[-Negate] <Boolean>

結果を無効にするかどうかを指定します。

[-RegEx] <Boolean>

フィルターテキストが正規表現かを指定します。

[-FilterName] <String>

フィルターに使用するテキストです。テキストは二重引用符で囲むことを推奨します。テキストが正規表現の場合は特に推奨します。

Limit（制限）

Function タブの結果件数を制限するか、または Function タブのすべての結果を表示します。

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Limit <Boolean>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

Limit <Boolean>

表示結果件数を制限するかどうか指定します。

Show Selected（選択関数を表示）

「関数」タブで選択関数のみを表示するように設定します。

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -ShowSelected <Boolean>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

ShowSelected <Boolean>

選択関数のみを表示するかどうかを指定します。

By Core（コアによるフィルター）

現在表示可能な結果の中から指定コアからの結果のみ表示するようにフィルター設定します。

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Core <Object>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

Core <Object>

整数のコア番号を指定してそのコアのみを表示するか、全ての文字列を指定してすべてのコアを表示するかを指定します。

By Thread（スレッドによるフィルター）

現在表示可能な結果の中から指定スレッドからの結果のみ表示するようにフィルター設定します。

```
PROFILER-FILTER [-Tab] <ScriptingTabs> -Thread <String>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

Group <String>

フィルターを設定するグループ名。すべてのグループを表示する場合は All を指定します。

例

```
PROFILER-FILTER Functions $false $true "^OS"
PROFILER-FILTER Functions -Negate $false -RegEx $true -FilterName
 "^OS"
PROFILER-FILTER Functions -ShowSelected $true
PROFILER-FILTER Threads -Core 0
PROFILER-FILTER Functions -Thread "ThreadName"
PROFILER-FILTER Counters -Group "GroupName"
```

22.2.6 PROFILER-INSTRUMENT

指定した関数名を基に計装対象の関数を設定します。

関数を設定するにはプロファイラ GUI がシンボルを認識している必要があります。計装プロファイルを取得する前に、簡易的なサンプルプロファイルを取得することによって簡単に認識させることができます。

```
PROFILER-INSTRUMENT [-FunctionName] <String>
```

[-FunctionName] <String>

プロファイラに表示される関数名と正確に一致している必要があります。C++ を使用している場合は、引数およびテンプレート情報もここに含まれます。

例

```
PROFILER-INSTRUMENT "TestFunction"
```

22.2.7 PROFILER-MARKED

マーカー付きコードをプロファイルデータ内に記録するかどうか指定します。

各タブにおいて GUI でこの値を個別に設定することが可能ですが、このコマンドレットはサンプルプロファイルおよび計装プロファイルの両方でマーカー付きコードを記録するようにフラグを立てます。

```
PROFILER-MARKED [-ShowMarkedCode] <Boolean>
```

[-ShowMarkedCode] <Boolean>

マークされたコードをプロファイルデータ内で記録するかどうか指定します。

例

```
PROFILER-MARKED $false
```

22.2.8 PROFILER-OPEN

NPROF ファイルを開きます。

```
PROFILER-OPEN [-FileName] <String>
```

[-FileName] <String>

開く対象の NPROF ファイルのフルパスです。このパス名は二重引用符で囲むことを推奨します。

例

```
PROFILER-OPEN "C:/Profiles/SavedProfile1.nprof"
```

22.2.9 PROFILER-PERFCOUNTERS

サンプルプロファイルおよび計装プロファイルの両方で、どのパフォーマンスカウンタを記録するかを設定します。

GUI でこれらの設定を個別に行うことが可能ですが、このコマンドレットを使用して両方同時に設定することができます。

```
PROFILER-PERFCOUNTERS [-PerfCounters] <ScriptingPerfCounters>
```

[-PerfCounters] <ScriptingPerfCounters>

パフォーマンスカウンタの詳細については、パフォーマンスカウンタのスクリプト実行を参照してください。

例

```
PROFILER-PERFCOUNTERS Disabled
```

22.2.10 PROFILER-SAMPLE

サンプルプロファイルにサンプリングストラテジーおよびサンプリングレートの両方を設定します。

```
PROFILER-SAMPLE [-Strategy] <ScriptingSampleStrategy> [-Rate] <Integer> [-NoWarning]
```

[-Strategy] <ScriptingSampleStrategy>

ストラテジーの値の詳細については、サンプリングストラテジーのスクリプト実行を参照してください。

[-Rate] <Integer>

レートの値の詳細については、レートを参照してください。

[-NoWarning]

このフラグは不正なストラテジーおよびレートの組み合わせがある場合に警告が発生しないようにします。

戻り値

ストラテジーおよびレートが正確に設定されているかを示すブール値です。

例

```
PROFILER-SAMPLE Time 1000
$success = PROFILER-SAMPLE -NoWarning -Strategy Time -Rate 100
```

22.2.11 PROFILER-SAMPLEGRAPH

サンプルグラフのオプションを制御します。このオプションは主にクィックフィルターで使用します。

ShowCombined（複合表示）

複合列の表示をするかどうかを制御します。

```
PROFILER-SAMPLEGRAPH -ShowCombined
<Scripting.SampleGraph.CombinedLineDisplay>
```

-ShowCombined <Scripting.SampleGraph.CombinedLineDisplay>

複合列の表示方法を設定します。複合列表示オプションの詳細については、SampleGraph.CombinedLineDisplay のスクリプト実行 を参照してください。

SampleDisplay（サンプル表示）

サンプルの表示方法を制御します。

```
PROFILER-SAMPLEGRAPH -SampleDisplay <Scripting.SampleGraph.SampleDisplay>
```

-SampleDisplay <Scripting.SampleGraph.SampleDisplay>

サンプルの表示方法を設定します。サンプルの表示オプションの詳細については、SampleGraph.SampleDisplay のスクリプト実行 を参照してください。

NameDisplay（名前表示）

名前の表示方法を制御します。

```
PROFILER-SAMPLEGRAPH -NameDisplay
<Scripting.SampleGraph.FunctionNameDisplay>
```

-NameDisplay <Scripting.SampleGraph.FunctionNameDisplay>

関数名の表示方法を設定します。関数名の列表示オプションの詳細については、SampleGraph.NameDisplay のスクリプト実行 を参照してください。

例

```
PROFILER-SAMPLEGRAPH -ShowCombined Only
PROFILER-SAMPLEGRAPH -SampleDisplay Mixed
PROFILER-SAMPLEGRAPH -NameDisplay Short
```

22.2.12 PROFILER-SAVE

現在のプロファイルデータを保存します。

```
PROFILER-SAVE [[-FileName] <String>]
```

[[-FileName] <String>]

任意です。保存対象ファイルのフルパスです。このパス名は二重引用符で囲むことを推奨します。値が渡されない場合はファイル名が日付となるファイルが現在のディレクトリに保存されます。

例

```
PROFILER-SAVE
PROFILER-SAVE "C:/Profiles/SavedProfile1.nprof"
```

22.2.13 PROFILER-SELECT

指定タブの項目を選択します。

TopGroup（上位グループ）

一度に各グループ 45 件の要素を選択します。

```
PROFILER-SELECT [-Tab] <ScriptingTabs> -Top [[-TopGroup] <Integer>]
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

-Top

一度に各グループ 45 件の要素を選択します。

[[-TopGroup] <Integer>]

選択対象のグループ。未指定の場合は、上位 45 件が選択されます。

List（一覧）

タブから指定したインデックスを選択します（インデックスは1番上が 0 で上から下に並んでいます）。

```
PROFILER-SELECT [-Tab] <ScriptingTabs> -List <Integer[]>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください

-List <Integer[]>

選択対象のインデックスの一覧。

DeselectAll（すべて選択解除）

現在選択されている要素のすべてを選択解除します。

```
PROFILER-SELECT [-Tab] <ScriptingTabs> -DeselectAll
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

-DeselectAll

タブの選択中の要素のすべてを選択解除します。

例

```
PROFILER-SELECT Functions -Top
PROFILER-SELECT Functions -Top 3
PROFILER-SELECT Functions -List 0 1 4 5
PROFILER-SELECT Functions -DeselectAll

[int[]]items = 0, 3
PROFILER-SELECT Functions -List @items
```

22.2.14 PROFILER-SHOWTAB

タブを表示します。

```
PROFILER-SHOWTAB [-Tab] <ScriptingTabs>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

例

```
PROFILER-SHOWTAB SampleGraph
```

22.2.15 PROFILER-START

指定したタイプのプロファイルの取得を開始します。

```
PROFILER-START [-ProfileType] <ScriptingProfileType>
```

[-ProfileType] <ScriptingProfileType>

プロファイルタイプの値の一覧についてはプロファイルタイプのスクリプト実行を参照してください。

例

```
PROFILER-START Sampled
PROFILER-START Instrumented
```

22.2.16 PROFILER-STOP

プロファイルのセッションを停止します。

```
PROFILER-STOP
```

例

```
PROFILER-STOP
```

22.2.17 PROFILER-SYNC

指定した ELF を使用してランタイムアプリケーションと接続します。

自動化のためには本機能を使用するより、一般的にはコマンドラインオプションを使用して ELF を自動的に読み込む指定方法を推奨します。

```
PROFILER-SYNC [[-FileName] <String>]
```

[[-FileName] <String>]

任意です。読み込み対象の ELF へのフルパスです。このパス名は二重引用符で囲むことを推奨します。ELF が具体的に指定されていない場合は、GUI が現在使用中の ELF を判定しそれを使用するようにします。

戻り値

接続が成功したかどうかのブール値です。

例

```
PROFILER-SYNC
$success = PROFILER-SYNC "C:/Game/bin/NDEBUG/game.elf"
```

22.2.18 PROFILER-TREECONTROL

指定タブで表示されているツリー上で操作を行うようにプロファイラに指示します。

```
PROFILER-TREECONTROL [-Tab] <ScriptingTabs> [-Action]
<ScriptingTreeControlActions>
```

[-Tab] <ScriptingTabs>

タブ名の一覧については タブのスクリプト実行 を参照してください。

[-Action] <ScriptingTreeControlActions>

ツリー上で適用する操作を指定します。利用可能な操作項目はツリー操作アクションのスクリプト化で説明しています。

例

```
PROFILER-TREECONTROL CallTree ExpandAll
PROFILER-TREECONTROL -Tab Info -Action CollapseAll
```

22.2.19 PROFILER-UNITS

どの単位を GUI に表示するかどうかを指定します。

```
PROFILER-UNITS [-UnitType] <ScriptingUnitType> [-ShowError] <Boolean>
```

[-UnitType] <ScriptingUnitType>

ユニットタイプの値の一覧については 単位タイプのスクリプト実行を参照してください。

[-ShowError] <Boolean>

エラー値を表示するかどうかを指定します。

例

```
PROFILER-UNITS Percent $false
```

22.2.20 PROFILER-UNSYNC

ランタイムアプリケーションから GUI を非接続にします。

```
PROFILER-UNSYNC
```

例

```
PROFILER-UNSYNC
```

22.2.21 PROFILER-WAIT

プロファイラを指定秒間待機させます。

> 補足：このコマンドレットの機能は内蔵の SLEEP コマンドと同等です。

```
PROFILER-WAIT [-Seconds] <Double>
```

[-Seconds] <Double>

待機時間（秒）です。分数値も可能です。

例

```
PROFILER-WAIT 0.5
```

22.2.22 PROFILER-WAITFOR

プロファイラが実行を続行する前に特定イベントを待つように指示します。

```
PROFILER-WAITFOR [-WaitEvent] <ProfilerWaitEvents>
```

[-WaitEvent] <ProfilerWaitEvents>

イベント値の一覧についてはプロファイラ待機イベントを参照してください。

例

```
PROFILER-WAITFOR ReadyToProfile
```

22.2.23 PROFILER-CTR-CORES

サンプルプロファイルで記録するコアをプロファイラに指示します。

```
PROFILER-CTR-CORES <Integer[]>
```

<Integer[]>

スペースで区切られた記録対象のコア番号の一覧です。一覧にコア番号が表示されていない場合は、記録対象にはなりません。記録対象コアの指定は 1 回のみとします。コア番号の順番は問いません。

例

```
PROFILER-CTR-CORES 0 1

[int[]]cores = 0, 1
PROFILER-CTR-CORES @cores
```

22.2.24 PROFILER-CTR-INFERRED

プロファイラーに「推測」プロファイルを取得するよう指示します。

PROFILER-CTR-INFERRED

「推測」プロファイルの取得が不可能である場合、その代わりに、これはプロファイラーに「コールスタック」プロファイルの取得を設定します。

例

```
PROFILER-CTR-INFERRED
```

22.3 引数の値

特定のコマンドレットの引数で指定可能な値は以下のとおりです。

22.3.1 コピー拡張

コールツリーおよびコードカバレッジタブには 1 つ以上のコピー機能があります。

以下の各オプションを選択すると、対応タブが一覧表示されます。

- **All（すべて） --** タブ：CallTree、CodeCoverage。デフォルト値です。タブ上のすべてのデータをコピーします。
- **NotSeen（未検出）** - タブ：CodeCoverage。unseen（未確認）関数の一覧のみをコピーします。
- **Seen（確認済み）** - タブ：CodeCoverage。seen（確認済み）関数の一覧のみをコピーします。
- **Selected（選択）** - タブ：CallTree。コールツリーから選択された関数のみをコピーします。

22.3.2 プロファイラ待機イベント

スクリプトエンジンで待機可能なイベントです。

- CanSync
- ReadyToProfile

22.3.3 レート

すべてのサンプルストラテジーで指定できないレート値があります。ストラテジーが対応するレートを確認したい場合は、GUI を使用してください。0 が指定されている場合は、レートには選択されたストラテジーに対してデフォルト設定されている記録値が使用されます。

22.3.4 パフォーマンスカウンタのスクリプト実行

スクリプト内でパフォーマンスカウンタを示す文字列です。

これらの値は、パフォーマンスカウンタグループ で説明されているパフォーマンスカウンタグループに対応しています。

- Disabled
- InstructionMissAndBussContention
- InstructionICacheEfficiency
- MemComputePerf
- CacheMemPerf
- DCacheReadPerf
- DCacheWritePerf

- PressureLoadStoreQueue
- MispredictedBranch

22.3.5 プロファイルタイプのスクリプト実行

開始可能なプロファイルのタイプです。

- Sampled
- Instrumented

22.3.6 サンプリングストラテジーのスクリプト実行

スクリプト内のサンプリングストラテジーを示す文字列です。

これらの値はサンプリングストラテジーおよびレートボタンで説明されているストラテジーに対応しています。

- Time
- MissICache
- MissDCacheRead
- MissDCacheWrite
- BranchMispredict
- StalledInstruction
- StalledDHazard
- StalledLSUFull

22.3.7 タブのスクリプト実行

特定のスクリプトコマンドに対する対象タブの一覧です。

- Assembly
- CallTree
- CodeCoverage
- Counters
- Functions
- Info
- Instrumented
- Last
- Threads

22.3.8 ツリー操作アクションのスクリプト化

ツリー上で実行可能な操作の一覧です。

- CollapseAll
- ExpandAll

22.3.9 単位タイプのスクリプト実行

プロファイラで表示される単位タイプです。

- AvgTime
- Percent
- Samples
- Time

22.3.10 SampleGraph.CombinedLineDisplay のスクリプト実行

サンプルグラフにおける複合列の表示方法を制御します。

- Hide
- Show

- `Only`

22.3.11 SampleGraph.NameDisplay のスクリプト実行

サンプルグラフにおける関数名やカウンタ名の表示方法を制御します。

- `Hide`
- `Short`
- `Long`
- `Full`

22.3.12 SampleGraph.SampleDisplay のスクリプト実行

コールスタックサンプルの表示方法を制御します。

- `Self`
- `Total`
- `Mixed`

22.3.13 スパイク検出タイプ

スパイク検出タイプの一覧は スパイク検出を参照してください。

- `Rare`
- `Burst`
- `Flutter`
- `BadFrame`

22.3.14 ストーリータイプ

ストーリー分析タイプの一覧は ストーリー分析を参照してください。

- `Brief`
- `Full`
- `File`
- `Top`

22.4 その他の有用なコマンド

スクリプティングの有用性を最大限に生かすため、役に立つと思われる追加のプロパティ、コマンド、関数呼び出しを以下に説明します。

- `[Environment]::CurrentDirectory`

  プロファイラアプリケーションのカレントディレクトリを取得します。通常は EXE が起動したフォルダになります。
- `[Nintendo.CPU.Profiler.Scripting.ScriptingCore]::NProfFilename`

  現在読み込んでいる NPROF ファイルのファイル名です。NPROF が読み込まれていない場合、またはまだ保存されていない新規プロファイルがある場合は、空文字列になります。
- `[Nintendo.CPU.Profiler.Scripting.ScriptingCore]::ScriptDirectory`

  現在実行中のスクリプトが読み込まれたディレクトリです。
- `[Nintendo.CPU.Profiler.Scripting.ScriptingGUI]::ELFDirectory`

  現在接続されている ELF のディレクトリです。

23　ニンテンドー 3DS CPU プロファイラゲーム API

ニンテンドー 3DS CPU プロファイラゲーム API は、プロファイル対象のゲームまたはアプリケーションに、プロファイルデータの強化する方法を提供します。最も重要な機能として、プロファイラがフレームベースの分析を実行できるように、ゲームが各フレームの開始点をマークできます。

API は C 言語で記述されています。いずれの関数を使用しても、「計装プロファイル」の取得が可能になります。

同梱の 関数リファレンス に API マニュアルが用意されています。

24 トラブルシューティング

本章では既知の問題点と対応策をまとめてあります。

24.1 プロファイラがクラッシュする

プロファイラがクラッシュした場合、ニンテンドー CPU プロファイラの実行可能ファイルと同じ場所にログファイルが保存されます。この場合、他の重要な情報と共にこのログファイルを任天堂に送付してください。迅速に問題対処にあたります。また、クラッシュの発生したバージョンの AXF ファイルと CCI ファイルのを保存しておいてください（問題の原因究明に役立つことがあります）。

24.2 サンプルグラフの描画が非常に遅く、フレームマーカーが重複する

ハートビートの記録が速すぎると（約 1 ミリ秒毎以上）、「サンプルグラフ」はすべての描画を試みるのため、描画スピードが極めて遅くなります。この場合はコード内のエラーを検索し、記録頻度を修正してください。

24.3 サンプルグラフを表示できない

サンプルグラフが表示されずにエラーメッセージが表示される場合は、そのエラーメッセージの内容に従って下さい。DirectX の要件の詳細は必要なシステムを参照してください。

24.4 サンプルグラフの表示が 1 フレーム遅れる

この問題が発生する場合は、使用中のグラフィックスドライバが正しくない、またはグラフィックスドライバが古い可能性があります。PC とグラフィックスカードに正しいグラフィックスドライバがインストールされているかどうか、および最新のグラフィックスドライバがインストールされているかを確認してください。

24.5 開発機材上でプロファイラが起動しない

プロファイラが開発機材上で起動できないと報告されることがあります。これが連続して発生してもアプリケーションが正常にスタートする場合は、まず開いているプロファイラアプリケーションすべてを終了させてください。再実行することで次回接続時に開発機材と正常に接続できるようになります。この問題が再び発生した場合、上記再実行を繰り返してください。この際にはデバッガソフトウェアを閉じて、開発機材の電源を入れ直してください。

> 補足：プロファイラは 20 秒でタイムアウトします。アプリケーションの起動に 20 秒以上かかるときは、任天堂までお知らせください。タイムアウト値の変更も検討いたします。

24.6 プロファイラは起動するがプロファイリング対象のゲームがスタートしない

この問題が発生する場合は、たいていテストメニュー画面が表示されたままとなります。注意事項およびリソースを参照して、動作条件や制限事項を満たしているかを再度確認してください。

24.7 計装プロファイルが取得できない

計装処理が不可能な場合、計装関数テキストボックスに以下のメッセージが表示されます。「必要な API リファレンスが見つからないため計装プロファイルが利用できません。詳細はマニュアルを参照してください。」計装プロファイルの取得には、ニンテンドー 3DS CPU プロファイラゲーム API のうち少なくとも 1 つの関数を使用する必要があります。これにより計装に必要な API リファレンスが含まれるようになります。

24.8 「LoadRun ライブラリがみつかりません。」というエラーが発生する

「LoadRunライブラリがみつかりません。ニンテンドー3DS との接続はできません。」というエラーが発生する

この場合は、デバッガソフトウェアを再インストールしてください。プロファイラが loadrun_ctr.dll の検索するためには、環境変数 *KMC_PARTNER_CTRS* または *IS_CTR_DIR* が設定されていること、そしてデバッガインストールフォルダの正しい位置をポイントしている必要があります。このファイルがなかったり、環境変数が正しく設定されていないと、デバッガの再インストール作業でこの環境変数を復元して dll の存在を確認します。

24.9 プロファイル結果が意味をなさない

プロファイル結果が現実的でない場合、無効な AXF ファイルがエミュレーションメモリに現在ロードされている CCI から指定された可能性が考えられます。2 つのファイルをお互いに同期させてから、切断と接続を繰り返して、プロファイルを再取得してください。

しかし、ARM チップの ABI の場合、正確なコールスタックの作成は非常に困難です。プロファイラでは最善の処理を行いますが、それでも問題が発生する可能性もあります（例えば、詳細不明なスタック）。問題が頻繁に発生する場合は、さらに調査を行いますので弊社窓口までお問い合わせください。

24.10 サンプルレートが高いほどスレッドに時間がかかる

プロファイラのオーバーヘッドのため、一定時間後に呼ばれるスレッドは、サンプルレートが高いほど時間がかかるようにみえます。これは想定内の動作です。この動作が問題であれば、サンプリングレートを低くするか、プロファイル中はスレッドを無効にしてください。

24.11 HIO デーモンが PCCOM と干渉する

HIO デーモンが開発機材と通信する機能と干渉することがあります。この問題を解決するには、HIO デーモンを再起動してください。

24.12 関数の名前がデマングルされない

この問題の対応には、ARMCC コンパイラツールを再インストールしてください。プロファイラでは ARMCC コンパイラツールのインストール時に設定された特定の環境変数に基づいて、ライブラリ提供のコンパイラを使用して名前をデマングルします。コンパイラツールがプロファイラ実行ファイルと同じ場所にインストールされていない場合、関数名はデマングルされません。

25 既知の問題点

以下は既知の問題点です。

25.1 すべて非表示後、コールツリーのコアが消える

「コールツリー」で、すべて展開してから一番下までスクロールしてすべて非表示を行うと、コア 0 とコア 1 が「コールツリー」から消えてしまうことがあります。また表示されているコアも、画面上部との間にギャップが生じて意図した表示より低くオフセットされてしまいます。この状況を回避するには、表示されているコアを展開するか、またはマウスのスクロールホイールを使用してスクロールダウンしてください。

25.2 $ 記号を含む関数のデマングル

$ 記号を含む関数名のデマングルに非対応の ARMCC 4.1 と 5.04 で提供されているデマングルに既知の制限があります。このように、これらの関数名は、いまだにプロファイラーでマングル化されて表示されます。

26 用語解説

バッファ

プロファイルデータ保存に使用する開発機材のメモリです。

コールツリー

各サンプルのコールスタックから構成される階層ツリーで、ゲームプロファイル中の関数呼び出しの構造を表示します。このツリーにはサイクルがないため、コールグラフとは若干異なります。コールツリーでは、同じ関数が呼び出し位置に基づき複数回表示されます。

コールスタック

スタック計装中に、ランタイムで発見された関数呼び出しの形跡です。

カウンタ

CPU カウンタ（パフォーマンスカウンタ）あるいはカウンタの種類 で説明されているその他のカウンタからのデータです。

ハートビート

特定のコアの周期的イベントです。プラットフォーム API のビルド関数を使用して手動で追跡可能です。ハートビートはフレームマーカーと類似していますが、どのようなタイプの周期的イベントとして定義可能なため、より一般的な用語として使用できます。ハートビートのマーカーは「サンプルグラフ」で確認できます。

ジッター

サンプリング時間のオフセットで、コード内でサンプル対象セクションを分散させます。適切なジッターが設定されていないと、固定サンプリングレートのため、コード内の特定セクションのみに集中してサンプリングが実行されてしまいます。

パフォーマンスカウンタによるサンプル

特定のパフォーマンスカウンタを使用して関数をサンプリングします。例えば、L1 データキャッシュミスが 500 回あるごとにサンプリングするというように選択することができます。ミスが 500 回に達するごとに、プロファイラはゲームが実行中の関数をサンプリングします。

時間によるサンプリング

指定したレートに基づいて関数をサンプリングします。

サンプリング

定期的にゲームを停止し、実行中の関数やコールスタックを記録します。

サンプリングレート

サンプル取得時に使用するレートです。

サンプリングストラテジー

サンプルの取得時間を決定するために使用する戦略です。

セルフ

ある関数のみに要した時間で、その関数が呼び出した関数の時間は含まれません。

サブ

ある関数に呼び出された関数（複数）の実行時間です（サブ = 合計 – セルフ）。

接続

開発機材に接続し ELF ファイルを読み込みます。プロファイラが接続された時点で、プロファイルの取得が可能になります。

合計

ある関数とその関数が呼び出したすべての関数の合計実行時間です。

単位

1つの関数における所要時間を示す単位です。表示単位には「パーセント」、「時間」、「サンプル」があります。

切断

開発機材との接続を切断します。

27 改訂履歴

版	改訂日	改訂内容
4.04	2015/04/22	「最後の関数」タブを追加しました。 Diff 機能と「関数」タブを追加しました。 新規機能用のスクリプトコマンドを追加しました。
		推測コールスタックプロファイルを取得する機能を追加しました。 インストールの手順をより明示的にしました。
4.03.2	2015/02/13	「CTR 下位互換性」ボタンを「強制互換」ボタンに変更しました。
4.03	2014/10/22	新規のスクリプトコマンド PROFILER-TREECONTROL を追加しました。 Quick Ribbon バーに詳細を追加しました。 関数詳細ウィンドウ操作に関するセクションを追加しました。 アセンブリタブの右クリック コンテキストメニューに関する情報を追加しました。 プロファイラの読み込み後、または取得後に Run Checkers が常に実行されるオプションを削除しました。 Debug Output タブを Console タブに変更しました。 サンプルグラフの右クリックコンテキストメニューの一部オプションの名前を変更しました。
		'SYSTEM WAITING' という通知を 'System Idle Thread' に変更。 リソースセクションを更新。
4.02.1	2014/08/06	設定可能なチェッカー機能項目を追加。 テキストフォーマット選択をデバッグ出力に追加。 関数選択コンテキストメニューをコード範囲に追加。
4.02.0	2014/07/30	チェッカー機能を追加。 関数詳細情報を追加。 スタックデプスグラフを追加。 Icicle 選択グラフを追加。 コンテキストメニューを追加。 アセンブリタブ項目の並び替え機能を追加。
		ニンテンドー 3DS におけるワークフローに関する注意事項を追加。 動作条件を更新。 誤ったツールを参照していた部分を修正。

版	改訂日	改訂内容
4.01.1	2014/05/06	Debug Output タブに関する記述を追加。
4.01.0	2014/04/16	レイアウトを統一。 情報タブに推測ハートビートのセクションを追加。 コマンドレット一覧を更新。 用語集を更新。 1st、2nd、3rd ボタンを Select Top ボタンに変更。 誤ったプラットフォーム情報を参照していた部分を修正。 正規表現のクイックフィルター一覧を削除。 コマンドラインの --loadelf オプションが CCL ファイルで動作しない既知の問題点を削除。
4.00.0	2014/02/28	新規の形式に変更。 Heartbeats タブを追加。 Quick Filter ボタンを更新。 API ドキュメントを別途 man pages へ移動。 リングバッファへの記録機能を削除。
3.01.8	2013/12/17	「.NET 4.0 ダウンロードリンクから Download をクリックしてください」という記述を削除。 CCL ファイルおよび --loadaxf コマンドラインオプションに関する既知の問題点を追加。
3.01.7	2013/12/13	.NET 4.0 ダウンロードページのリンクを更新。
3.01.6	2013/02/26	プロファイラが HIO 通信の排他制御を行う必要性に関する制限事項（セクション 1.4）を削除。
3.01.5	2013/02/19	対応 SDK を更新。
3.01.4	2012/10/29	対応ファームウェアを更新。
3.01.3	2012/09/06	既知の問題点を修正したため、14.4 プロファイリング中に開発機材がロックするを削除。
3.01.2	2012/08/24	修正済みの既知の問題点を削除。
3.01.1	2012/08/23	プロファイル対象のコアを選択するための新規の関数リファレンスを追加。
3.01.0	2012/08/14	改訂履歴をドキュメントの末尾に移動。 修正済みの CCL ファイルに関する既知の問題点を削除。 セクション 3.2.5 を追加。コア選択に関する情報を追加。 プロファイリングのヒントの章を新規追加。

版	改訂日	改訂内容
3.0.0	2012/05/07	nn::ro ライブラリを使用するアプリケーション（例：DLL を使用するアプリケーション）をプロファイルする機能が作動しない既知の問題点を削除。 IS-CTR DEBUGGER の CCL ファイル対応に関する既知の問題点を更新。 関数名のデマングルに必要な依存関係の変更に伴い 14.11 の説明を更新。 現在使用中の開発機材をより反映するように 1.3 の説明を更新。
2.01.0	2012-01-24	IS-CTR に関する説明を追加。 IS-CTR および CCL ファイルに関する既知の問題点を追加。 AXF ファイルの代わりに CCI／CCL ファイルを読み込むように記述を更新。
2.00.0	2011/11/30	SDK-2_X のファームウエアバージョンを更新。 SDK-3_X 対応に関する情報を追加。 PCCOM サーバーについての記述を削除。 プロファイラと PCCOM サーバーについてのトラブルシューティングの記述を削除。 既知の問題点「DLL を使用したアプリケーションのプロファイルを試みるとクラッシュする」を追加。
1.01.0	2011/10/19	バージョンを更新。 SDK-1_X についての記述を削除。
1.00.0	2011/06/22	製品名をニンテンドー 3DS CPU プロファイラに更新。 マニュアルボタンのセクションを追加。 システムリソース内メモリー使用量制限の誤記を修正。 制限事項とトラブルシューティングのセクションの相互参照をプロファイルの取得のセクションに追加。
0.14.2	2011/04/27	計装対象関数の制限に関する注意事項を更新。 既知の問題点「プロファイリング中に開発機材がロックする」を更新。
0.14.1	2011/03/18	計装対象関数の制限事項の追加。 複数の SDK バージョンに対応するためドキュメントの変更。プロット表示関数の削除。 既知の問題点から「プロファイリング停止時にアクセス違反が発生する」を削除。
0.14	2011/02/25	リバースコールツリーのセクションを追加。 サンプルグラフの描画時間に関するセクションを追加。 プロファイラのリソース使用量に関するセクションを追加。 SDK の制限に起因するスレッドの制限事項を削除。
0.13.1	2011/01/28	バッファサイズ変更の記述を更新。

版	改訂日	改訂内容
		サンプルグラフの章にアルファに関する説明を追加。
0.13.0	2011/01/21	計装タブの説明を更新。 計装グラフタブの説明を追加。 コードブロックを追加。 HOME メニューとの併用に関する記述を追加。 バッファサイズ変更に関する記述を更新。 計装プロファイルの関数選択の制限を変更。 バッファサイズ API 関数を削除。
0.12.1	2010/12/20	計装プロファイル取得条件の情報を追加。
0.12.0	2010/12/09	ARM 計装を追加。 GUI のレイアウトを修正。 パフォーマンスカウンタの章を追加。 トラブルシューティングおよび既知の問題点を更新。
0.11.0	2010/11/17	Units の記述に Error ボタンについての説明を追加。 1st、2nd、3rd、4th ボタンの説明を追加。 トップ 50 ボタンの説明を削除。 関数タブのフィルタリングについて追記。 Time ボタンの新規の動作を反映させるため説明を修正。 サンプルグラフにおける vblank の赤い縦線について追記。 サンプルグラフにおけるフレームレートボタンについての説明を追加。 Heatmap モードにおけるフレームレートボタンについての説明を修正。 自動フレーム検出補助機能（非推奨）を追加。 プロファイラゲーム API を一部変更。動作条件を更新。
0.10.1	2010/10/14	API 宣言における不具合を修正。 EnableProfiling、DisableProfiling 関数を追加。 C++ API に関するリファレンスを削除。
0.10.0	2010/10/07	ヒートマップ機能の説明を追加。 パフォーマンスカウンタによるサンプリング機能の説明を追加。 ランタイムコントロール関数のリファレンスを追加。 コマンドラインのリファレンスを追加。 サンプリングレートの説明を更新。
0.09.1	2010/09/10	インストールの手順を更新。 PARTNER デバッガソフトウェアのバージョンを 5.61-091-20100901 に変更。

版	改訂日	改訂内容
		複数の開発機材についての不具合や注意事項を削除。 UNKNOWN STACK DETAILS の原因を更新。 トラブルシューティングを更新。
0.09.0	2010/09/03	パフォーマンスカウンタタブの説明を追加。 制限事項に関するセクションを追加。 プロファイル開始の手順を変更。 Count ボタンを Samples ボタンに変更。 Time の説明に精度に関する警告を追加。
0.08.0	2010/08/06	インストールの手順に .nprof ファイルをプロファイラに関連付ける方法を追加。 スレッド選択の項に、アクティブに関する説明を追加。 Top 50 ボタンの説明を追加。 スレッドボタンおよびスレッドヘッダに関する説明を追加。 SYSTEM WAITING の説明を追加。 情報タブ内のプロファイル統計における新規項目を追加。 フィルタリングについて文言を修正。
0.07.0	2010/07/27	初版

記載されている会社名、製品名等は、各社の登録商標または商標です。