クラス、関数、マクロ : wxDebugContext, wxObject, wxLog, Log functions, Debug macros

アプリケーションのデバッグに役立つ様々なクラス、関数、マクロが wxWidgets で準備されている。これらのうちの大部分は、wxWidgets、アプリケーション、wxWidgets を使用している全てのライブラリの全てにおいて __WXDEBUG__ シンボルを定義してコンパイルしたときだけ有効になる。デバッグモードでのみ有効であるようなコードを実行するために自身のアプリケーションで __WXDEBUG__ シンボルを検証することができる。

■ wxDebugContext

wxDebugContext は、インスタンスを持つことは無く、様々な静的関数と静的変数を持っている。全てのオブジェクトをストリームにダンプしたり、確保したオブジェクトに関する統計を書き出したり、メモリエラーをチェックすることができる。

wxDebugContext::Dump が wxObject::Dump を呼び、アプリケーションの状態に関する重要な情報を与えるために、wxWidgets クラスから派生した各クラスの wxObject::Dump メンバ関数を定義することは良い慣習である。

もしメモリリークの調査に苦労しているのであれば、適当な場所で wxDebugContext::Dump と wxDebugContext::PrintStatistics を呼び、デバッグモードでリコンパイルすると良い。これらの関数はまだ削除されていないオブジェクトが何であるか、それらはどんな種類のオブジェクトなのかを教えてくれる。事実 wxWidgets は、デバッグモードでは、アプリケーションが終了しようとしているときにメモリリークを自動的に検出し、何らかのリークがあれば、wxWidgets はその問題に関する情報を提供する（どのくらいの情報かは、OS やコンパイラに依存する。あるシステムでは、全てのメモリログの取得が許されていない）。使用例として、memcheck サンプルを参照のこと。

wxDebugContext が動作できるよう、動的に確保されたオブジェクト（静的に宣言されたオブジェクトではなく）に関する特別な情報を保存するために wxObject の new 演算子と delete 演算子が再定義されている。これによりデバッグバージョンでのアプリケーションの動作は遅くなるが、検出が困難なメモリリーク（オブジェクトの未開放）、オーバーライト（オブジェクトのサイズを超えたメモリの書き換え）、アンダーライト（オブジェクトの前のメモリの書き換え）を検出することができる。

デバッグモードが有効な場合、シンボル wxUSE_GLOBAL_MEMORY_OPERATORS が setup.h で 1 に設定され、'new' は以下のように定義される。：

```
#define new new(__FILE__,__LINE__)
```

wxWidgets とアプリケーションでの全ての 'new' は、２つの特別な引数を持つようにオーバーライドされた演算子で置き換えられる。これは、デバッグ出力（及び、メモリ問題をレポートするエラーメッセージ）はどのファイルのどの行でオブジェクトが確保されたのかを知ることができることを意味する。残念ながら、全てのコンパイラが適切に動作するように定義する訳ではないが、大部分はうまくいく。

■デバッグマクロ

できるだけ早い時点で､問題点に対して検証するための wxASSERT をコードの中に自由に埋め込むことにより､' 保守的なプログラム ' 手法の一部分として、デバッグマクロを使用することもできる。前向きに考えて、長い目で見れば、驚くべき多くの時間を節約できる。

wxASSERT は、条件が true で無い場合にエラーメッセージボックスを表示する。自分自身のエラーメッセージを提供するために、wxASSERT_MSG を使用することができる。例：

```
void MyClass::MyFunction(wxObject* object)
{
    wxASSERT_MSG( (object != NULL), "object should not be NULL in MyFunction!" );

    ...
};
```

メッセージボックスは、プログラムを続けて実行するか、中断するかを決めることができる。もしデバッガ上でアプリケーションを実行しているのであれば、問題が発生している箇所を正確に知ることができる。

■ログ関数

デバッグモードで、デバッグのための情報を出力するために､wxLogDebug と wxLogTrace を使用することができる。それは、非デバッグコードでは無効になる。

wxDebugContext 概要

クラス : wxDebugContext

wxDebugContext は、様々なデバッグ機能とメモリトレース操作を実行するためのクラスである。

この関数は、静的メンバデータと静的メンバ関数のみを持っており、インスタンスは持たない。恐らく､最も有用なメンバは､SetFile（デフォルト標準エラーやデバッガ出力の代わりに、ファイルに出力するため)、Dump（動的配置オブジェクトのダンプ)、そして、PrintStatistics（オブジェクトの配置に関する情報のダンプ）であろう。また､メモリブロックの完全性 (integrity) をチェックするために Check を使用することができる。

以下は、使用方法の例である。SetCheckpoint は、ダンプされる箇所をそれ以降に限定するためのチェックポイントを設定する。

```
wxDebugContext::SetCheckpoint();

wxDebugContext::SetFile("c:\\temp\\debug.log");

wxString *thing = new wxString;

char *ordinaryNonObject = new char[1000];

wxDebugContext::Dump();
wxDebugContext::PrintStatistics();
```

__WXDEBUG__ が定義されているか、あるいは他の場合（setup.h で wxUSE_DEBUG_CONTEXT が 1 に設定されている場合）に wxDebugContext が使用できる。エラーログ機能を使用するためだけに、wxWidgets やアプリケーション全体をリコンパイルしなくてもよいよう、非デバッグ時には無効にならない。

注意：今のところ、wxDebugContext::SetFile には問題がある。そのため、代わりにデフォルトストリームを使用すること。最終的には、ロギングは、wxLog 機能を使用することになる。