DLL 呼び出し

プラットフォーム呼び出しサービス

.NET プラットフォームは、一つのシステムとして完成しています
その仕組みは、GUI や ネットワーク環境など、システムとして独立できる能力があり
.NET 標準クラスライブラリを使えば、GUI アプリケーションを作成することも簡単です

しかし、.NET はネイティブプラットフォームの機能を呼び出す手段も提供しています
Win32 API プログラマは、独自の DLL を C/C++ で作成する能力があるでしょう
.NET からプラットフォームを呼び出すことができれば、便利なのは間違いありません
ただし、システムを呼び出した場合 .NET アプリケーションはそのシステムに依存してしまいます

DLL を呼び出すことができれば、事実上 Windows の全ての機能を引き出すことができます
しかし、.NET の思想は OS を抽象化することであることを忘れてはいけません
.NET におけるシステム呼び出しサービスを **PInvoke** と呼ぶこともあります

システムを呼び出すには、コンパイラに API の位置を知らせる必要があります
これには、**System.Runtime.InteropServices.DllImport** 属性を用います

public sealed class DllImportAttribute : Attribute

この属性クラスのコンストラクタは次のように定義されています
コンストラクタ引数は、位置パラメータとして指定しなければなりません

public DllImportAttribute(string dllName);

dllName には、DLL の名前を指定します
DLL の本体は、システムが発見できる位置になければなりません
この属性が指定されたメソッドは、指定した DLL に存在すると解釈されます

Windows API は C 言語から呼び出せるように設計されていて
オブジェクト思考の立場から見れば、それは常に public static なメソッドです
呼び出す API のエントリポイントは、コンパイラがメソッド名から判断します

また、C# 言語では外部メソッドを呼び出す場合、
必ず、メソッドに **extern** 修飾子を指定しなければなりません
この extern キーワードは、一般的に DllImport 属性とセットで用いられます

どの DLL を呼び出すかは、プログラマが判断する作業です
DLL の知識、及び Win32 API のネイティブな型情報を知っている必要があります

```
using System;
using System.Runtime.InteropServices;

class Win32 {
 [DllImport("USER32.DLL")]
 public static extern int MessageBoxA(
  int hWnd , String lpText , String lpCaption , uint uType
 );

 public const int MB_OK =  0x00000000;
 public const int MB_OKCANCEL =   0x00000001;
 public const int MB_ABORTRETRYIGNORE = 0x00000002;
 public const int MB_YESNOCANCEL =  0x00000003;
 public const int MB_YESNO =   0x00000004;
 public const int MB_RETRYCANCEL = 0x00000005;

 public const int MB_ICONHAND =   0x00000010;
 public const int MB_ICONQUESTION = 0x00000020;
 public const int MB_ICONEXCLAMATION = 0x00000030;
 public const int MB_ICONASTERISK =  0x00000040;
}

class Test {
 public static void Main() {
  Win32.MessageBoxA(0 , "Kitty on your lap" ,
   "MsgBox" , Win32.MB_YESNO | Win32.MB_ICONEXCLAMATION);
 }
}
```

このプログラムは、Win32 の MesssgeBox() ファンクションを呼び出します
プログラムが実行されると USER32.DLL がメモリにロードされ
API が呼び出され、Kitty on your lap と書かれたダイアログが出現します

DllImport 属性が指定された Win32.MessageBoxA() メソッドは
USER32.DLL ライブラリの MessageBoxA() というエクスポート関数であることを表しています

DllImport の EntryPoint 名前付きパラメータを使用すれば
メソッド名ではなく、このパラメータで DLL の関数の位置を指定することができます
こうすれば、独自の新しい名前で DLL の関数を C# から呼び出すことができるでしょう

```
using System;
using System.Runtime.InteropServices;

class Win32 {
 [DllImport("USER32.DLL" , EntryPoint="MessageBoxA")]
```

```
 public static extern int Dialog(
  int hWnd , String lpText , String lpCaption , uint uType
 );

 public const int MB_OK =  0x00000000;
 public const int MB_OKCANCEL =   0x00000001;
 public const int MB_ABORTRETRYIGNORE = 0x00000002;
 public const int MB_YESNOCANCEL =  0x00000003;
 public const int MB_YESNO =   0x00000004;
 public const int MB_RETRYCANCEL = 0x00000005;

 public const int MB_ICONHAND =   0x00000010;
 public const int MB_ICONQUESTION = 0x00000020;
 public const int MB_ICONEXCLAMATION = 0x00000030;
 public const int MB_ICONASTERISK =  0x00000040;
}

class Test {
 public static void Main() {
  Win32.Dialog(0 , "Kitty on your lap" ,
   "MsgBox" , Win32.MB_YESNO | Win32.MB_ICONEXCLAMATION);
 }
}
```

このプログラムの Win32.Dialog() メソッドは
DllImport 属性で USER32.DLL の MessageBoxA() 関数を呼び出すことを表しています
プログラムを実行すれば、先ほどのプログラムとまったく同じ結果を得ることができます

ところで、C# の int 等の数値型は、単純な数値型ではありません
この実体は構造体ですし、String 型も単純な 1 バイトの配列とは異なります
Win32 API において、文字列とは単純な配列であり、クラスや構造体ではありません
C# から DLL を呼び出す場合、このギャップはどのように解決しているのでしょうか

実は、.NET は**マーシャリング**と呼ばれる方法でこれを解決します
.NET 型を DLL 関数に直接渡すことは、互換性がないためできません
そこで、.NET は DLL を呼び出す時、マーシャリングと呼ばれる処理を行うことによって
.NET 型の値をネイティブ型に変換し、DLL に情報を渡しているのです

そもそも、.NET の型にはネイティブ型の情報が属性として含まれているため
String 型を DLL に渡そうとすれば、Win32 の LPSTR 型に変換される仕組みを持っているのです
より詳しくこの実装を見たければ、**MarshalAs** 属性を調べるとよいでしょう
この属性も System.Runtime.InteropServices 名前空間に含まれています

前のページへ　　戻る　　次のページへ