特集2

業務に役立つスレッドプログラミング

スレッドプログラミングを業務アプリケーションに活用するための基礎と応用

日本ユニシス株式会社
.NETビジネスディベロプメント
尾島 良司
OJIMA, Ryoji

文化とは無縁の人生を送っている、音楽も聴かなければテレビも見ない私ですが、実は子供の頃はピアニスト志望だったのです。忘れもしない小学校5年生のとき、手がでかいという理由で両親にピアニスト向きだと判断された私は、ピアノ教室の門を激しく叩いたのでした。3か月くらい後、バイエルの右手と左手をバラバラに動かすところで挫折しちゃいましたけど。そう、複数のことを同時に処理するのは非常に難しいことなのです。私が不器用なのではなく、このような基本的なことを無視したピアノの設計に問題があるのです。そしてこれはプログラミングでも同じ。同時に複数のことをさせる、つまりスレッドプログラミングには多数の落とし穴があるのです。というわけで、本稿では、このスレッドプログラミングについて考えてゆきます。

Technology Tools

- ❑ Visual Basic .NET
- ☑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ☑ ASP.NET
- ❑ Internet Information Services
- ❑ Other:

Level

★ ★ ★ ★ ★

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、付録CD-ROMの¥DOTNET¥F02ディレクトリに収録しています。

¥SRC
サンプルプログラムのソースコードとバイナリ

そもそも、スレッドとは？

「スレッドとは、プログラムカウンタとスタックのことである」

スレッドについて質問されると、いつもこのように答えちゃう私。

私と同世代の化石人間なら肌で理解できるこの説明ですが、初めて触わったコンピュータがWindowsという恵まれた人には、わけがわからないかもしれません。

少々遠回りではありますが、コンピュータの基本的な仕組みから始めさせてください。知っておいて損はないと思いますよ。

✚プログラムカウンタ

実は、コンピュータの心臓であるCPUは、一度にひとつのことしか実行できません。メモリからデータを読んだり、計算したり、メモリに書き出したり。これらの作業を逐次実行することで、コンピュータはデータベースからデータを検索したりDVDを再生したりしているのです。

そして、現在のコンピュータはノイマン型コンピュータです。ノイマン型コンピュータである以上は、メモリからプログラムを読み込んで実行します。命令をメモリからひとつだけ読み込んで実行、実行が終了したら命令を次のアドレスのメモリからひとつだけ読み込んで実行。これを繰り返すわけです。

ここまでをまとめると、コンピュータというのは、「メモリの中にある命令を順番に実行してゆくだけの機械」と定義できます。そして、順番に実行するためには現在どこまで実行したのかを表現する値が必要で、これを“プログラムカウンタ”と呼びます。プログラムカウンタが指し示すメモリにある命令を読み込む、読み込んだ命令を実行、そしてプログラムカウンタを“1”増やす。これを繰り返すことでコンピュータは仕事をこなしてゆくのです。

ただし、順次実行だけでは複雑な処理は記述できません。ifとかforとかwhile、メソッド呼び出しに相当するものが必要なのです。で、これらがどうやって実現されているかというと、先ほどのプログラムカウンタの値を操作することで実現されています。

✚スタックとヒープと静的領域

コンピュータはデータを処理する機械です。ノイマン型コンピュータでは、データはプログラムと同様にメモリに配置されます。

さて、このデータのメモリへの配置方法は一般に3つあります。“スタック”と“ヒ

ープ”と“静的領域”です。

スタックの例として、再帰で階乗を求めるプログラムを考えてみましょう。

階乗は、

n×(n - 1)×・・・×3×2×1

と定義されています。“5”だったら、5×4×3×2×1で“120”ですね。

さて、リスト1のint Factorial()が階乗を計算する関数なのですけど、Factorial()は内部で自分自身を呼び出しています。このように自分自身を呼び出すものを“再帰”と呼びます。

ここで気をつけてください。Factorial()の引数であるnumberは、Factorial()が呼ばれるたびに「n－1」「(n－1) －1」「((n－1) －1) －1」と変わってゆきます。同じnumberという変数だからと言って、このデータの格納場所に同じ領域を使用することはできません。「number * Factorial (number- 1)」としているのですから、Factorialを呼び出したためにnumberの値が変わってしまうのでは困るわけです。

というわけで、COBOLのような古い言語を除けば、関数やプロシージャの引数やその中で宣言された変数は、スタックに配置されることになっています。

スタックとは、最後のデータが先に出力されるという特徴を持つデータの格納方法です（そして、通常CPUは、スタックを扱うための専用の命令を持っています)。後に実行された関数のほうが先に終了するわけですから、関数のための情報を格納する方式としてはスタックが最適なのです。

では、ヒープとは何か？　クラスや配列をnewした際に確保されるエリアがヒープです。フィールドもヒープに格納されます。ヒープは関数とは関係のない単位で確保／解放されるので、newでメモリを確保して、ガベージコレクションによって解放されます。

残る静的領域というのは、staticな変数を格納する場所です。静的領域は解放とは無関係です。staticは、いつでもアクセスされる可能性があるわけですから。

リスト1：再帰を使って階乗を求める

```
using System;

public class List1
{
   public static void Main()
   {
      Console.WriteLine("5の階乗は{0}です。", List1.Factorial(5));
   }

   static int Factorial(int number)
   {
      return number == 0 ? 1 : number * Factorial(number - 1);
   }
}
```

✚で、スレッドとは？

スレッドとは、非常に短い期間でOSが実行するプログラムを切り替えることで、同時に複数の処理をしているかのように見せかける技術、またはその技術で実行されるプログラムの単位です。

スレッドの切り替えは、プログラムカウンタの値を変更することで実現できます。ただし、何も考えずにプログラムカウンタを変更してしまうと、元のスレッドに戻すことができなくなってしまいます。よって、切り替えの際には元のプログラムカウンタを覚えておきます。

また、プログラムカウンタを切り替えるだけでは、スレッドはまともに動作しません。関数やプロシージャの実行途中で別の関数やプロシージャに処理が移ってしまうのですから、スタックの先入れ後出しという機能では対応できないのです。そこでスレッドの数だけスタックを用意し、スレッドの切り替え時にはスタックも切り替えることにするわけです[注1]。

ね、スレッドって、プログラムカウンタ＋スタックでしょ？

スレッドは、使うだけならとっても簡単

さて、小難しい理屈は終わり。実際にスレッドプログラミングしてみましょう。

リスト2を見てください。リスト2は私の世代の人間なら泣いて喜ぶマンデルブロ集合を描くプログラムです（図1)。

マンデルブロ集合とかフラクタルとかカオスはさておいて、リスト2には起動が遅いという致命的な欠陥があります。マンデルブロ集合を計算する処理には数秒必要で、それが 終わるまでウィンドウが出てこないのです。実際に試してみてください。これではたまらないですよね？

まずウィンドウを表示することにして、バックグラウンドで計算をさせることにしましょう。Wordがバックグラウンドで印刷をするような感じです。スレッドはこのような場合に役に立ちます。とい

注1）.実際には、コンテキストを切り替えるわけですけど。

図1：マンデルブロ集合