R流・C#マルチスレッドの復讐

2009年05月16日

R・田中一郎

http://blogs.wankuma.com/rti/

Microsoft MVP for Development Tools - Visual C# (April 2007 - March 2010)

Microsoft Certified Professional Developer

- *Windows Developer*

Microsoft Certified Technology Specialist

- *.Net Framework 2.0: Windows Applications*

自己紹介

会員番号:34

名前:R･田中一郎

所在:栃木県

年齢:18才

職業:

主に業務用システムの開発

2005年11月
Microsoft Visual Studio .NET デビュー。
この頃から R.Tanaka.Ichiro と名のりネットでアクティブに活動を始める。

2006年02月
C# を学び始める。理想的な言語に感動。尊敬する方々の影響も大きい。

2006年09月
わんくま同盟加盟
ある事件がきっかけで、中さんから声をかけていただき加盟。

2006年11月
MSC2006 にて R･田中一郎として始めて人前に姿を晒す。

2007年04月
Microsoft MVP for Visual Developer - Visual C# を受賞。

2007年06月
わんくま同盟勉強会にてスピーカーデビュー。

2008年04月
Microsoft MVP for Development Tools - Visual C# を受賞。

2008年05月
70-526,70-536 試験をパス。
Microsoft Certified Technology Specialist for .Net Framework 2.0: Windows Applications 資格取得。

2009年04月
Microsoft MVP for Development Tools - Visual C# を受賞。

70-548 試験をパス
Microsoft Certified Professional Developer for Windows Developer 資格取得。

こんな経験はありませんか？＜Ｄｅｍｏ＞

処理中に固まる

クライアントのリクエストが同時に処理できない

CPUがフルパワーで使えない

通常の処理の流れ

処理1

↓

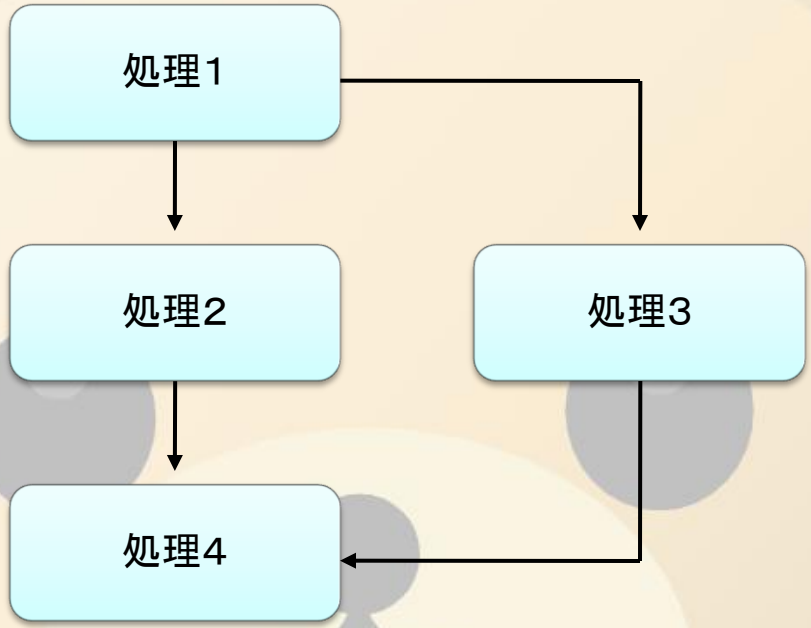
処理2

↓

処理3

ひとつずつ処理する

マルチスレッドな処理の流れ



複数の処理を並列に実行できる

BackGroundWorker

System.ComponentModel.BackgroundWorker

処理中にユーザー インターフェイスが応答しなくなることを防ぐために、処理を別スレッドで実行できる

＜Ｄｅｍｏ＞

Thread

System.Threading.Thread

最も、基本的な方法。
優先度の設定などの細かい制御ができる。

＜Ｄｅｍｏ＞

ThreadPool

System.Threading.ThreadPool

スレッド数を制御したり、スレッドリソースを再利用することで、パフォーマンスの良いマルチスレッドの処理ができる。

スレッドメソッドの引数を1つ取ることが可能。

＜Ｄｅｍｏ＞

Timer

System.Threading.Timer

一定間隔でスレッドを作成して処理を実行させることができる。

＜Ｄｅｍｏ＞

Delegate

スレッドプールを自動的に使用してメソッドを実行する。

戻り値の取得や、引数を複数使用できる。
また、例外処理や終了処理の待機も可能。

＜Ｄｅｍｏ＞

はまりどころ

•変数に格納される値が常時変化してしまう問題～排他制御(Lock)

•最適化によって必要な処理が省かれてしまう問題～最適化防止(volatile)

•非同期であるが故の問題
- 終わらなくても次の処理が動作してしまう
- 実行される順序がバラバラ

C#4.0 - Task

```
using System.Threading.Tasks;
 :
Task t0 = Task.StartNew(() => 重い処理(0));
Task t1 = Task.StartNew(() => 重い処理(1));
```

C#4.0 – for

通常

```
for (int i = 0; i < 10; ++i) {
    重たい処理(i);
}
```

並列

```
Parallel.For(0, 10, i => {
    重たい処理(i)
});
```

<Demo>

C#4.0 – foreach

通常

```
foreach(var x in collection) {
    重たい処理(x);
}
```

並列

```
Parallel.Foreach(collection, x => {
    重たい処理(x)
});
```

<Demo>

C#4.0 – LINQ

通常

```
var q =
   from x in collection select x;
```

並列

```
var q =
   from x in collection.AsParallel() select x;
```

<Demo>

まとめ

- 記述は簡単になって使いやすくなります
- でも、はまり所は今までと同じです
- 処理の内容を理解して用法用量をまもって
  正しく使いましょう

ご清聴ありがとうございました！

2009年05月16日

R･田中一郎

http://blogs.wankuma.com/rti/

Microsoft MVP for Development Tools - Visual C# (April 2007 - March 2010)

Microsoft Certified Professional Developer

- *Windows Developer*

Microsoft Certified Technology Specialist

- *.Net Framework 2.0: Windows Applications*