επιστημηの

オブジェクト指向的夜話

★C#から見た
★.NET Framework

第9回

コレクション二題

επιστημη
えぴすてーめー

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¥YAWAディレクトリに収録しています。

¥ALGORITHM
汎用の（線形）検索アルゴリズムサンプル

¥FUNCTIONAL
関係演算子からdelegateを作るサンプル

¥TRIAL
お試しアセンブリ

書籍のお話

新春早々C++関連の本が出ます（出ました、かな？）。『C++の設計と進化』（ISBN：4-7973-2854-1）というタイトルで、C++の産みの親Bjarne Stroustrupが10年前に書いた本、『The Design and Evolution of C++』の完全訳。訳はおなじみ岩谷宏さん、僕は監修をやらせていただきました。.NETでもC#でもない私事で恐縮ですけど、C#/Java/VB.NETな方々にも手にとってご覧頂きたく。と言うのも、C++は“なぜ多重継承を許すのか”、“なぜガベージコレクションをしないのか”などなど、C++の“なぜ”が設計者自身の言葉で語られているんです。内容はC++の深淵に迫りますが、翻せばC#/Javaが多重継承をしないのか／ガベージコレクションを持っているのか……、C#やJavaをはじめとした（おそらく）C++を父／母にする言語の“なぜ”が垣間見える内容かと思います（岩谷さんもあとがきで同様の感想を述べられてます）。特に今回の日本語訳のために、Dr.Bjarne自らの手による30ページ強の「第0章：2005年のC++」が追加され、原著刊行からこれまでの10年さらに未来のC++を語っていただきました。原著に先駆けてこんな素敵な加筆がなされるなんて、この世界では極めて稀なことです。書店で見掛けたらパラパラとめくっていただければ幸いに存じます（というか、買ったほうがいいと思う：編集部注）。

「辞書」のからくり

この本の監修作業に着手したころ、本連載の第4回（2004年10月号）原稿を書いてました。「コレクションで遊ぼ」と題した、.NET Frameworkの提供するコレクショ

リスト1：SortedList パフォーマンス計測

```
using System.Collections;

public class trial {

 /* 昇順に挿入 */
 public static int forward(int n) {
  IDictionary collection = new SortedList();

  int start = System.Environment.TickCount;
  for ( int i = 0; i < n; ++i ) {
   collection.Add(i,i);
  }
  return System.Environment.TickCount - start;
 }

 /* 降順に挿入 */
 public static int backward(int n) {
  IDictionary collection = new SortedList();

  int start = System.Environment.TickCount;
  for ( int i = n; i > 0; --i ) {
   collection.Add(i,i);
  }
  return System.Environment.TickCount - start;
 }

 public static void Main() {
  for ( int n = 1000; n < 100000; n *= 2 ) {
   System.Console.WriteLine(" forward({0}): {1}", n,
    forward(n));
   System.Console.WriteLine("backward({0}): {1}", n,
    backward(n));
  }
 }
}
```

ン（データの集合）の解説ですね。そこでは名前空間System.Collectionsにあるコレクションのパフォーマンス比較なんかをやらかしています。IDictionaryを実装したHashtableとSortedListとを比較したり。

○おさらい

ここでSortedListについてちょっとおさらい。SortedListの中身はArrayListすなわち可変長配列でできていると思ってよさそうです。ArrayListと異なるのは、コレクション内の各要素がいつもきちんと昇順に並ぶように要素の挿入が行なわれます。SortedListに要素を挿入するとき、小さい順（昇順）に挿入する場合と大きい順（降順）に挿入する場合との速度比較をやってみました（リスト1・図1）。その差は歴然、降順に挿入するとすっごく遅いのです。SortedListがArrayListで作られていると推察できます。つまり降順に挿入ということは、要素はいつもコレクションの先頭に挿入されます。配列の途中に要素を挿入するには、挿入位置より後ろにある要素をひとつずつズラして空席を確保しなければなりません。挿入位置が先頭ということは、挿入の度にコレクション内の全要素に対する移動が伴うわけですから遅いのも頷けます（逆に昇順であれば挿入位置は常にコレクションの末尾なので要素移動の必要はありません）。要素の削除についても同様です。コレクションの先頭に近い要素

図1：リスト1の実行結果

ほど、その削除にはそれより後ろの要素をひとつずつ前方にズラして空席を埋めなくてはなりません。SortedListへの要素の挿入／削除にはおおむね要素数に比例する時間を要します。

一方SortedListが保持する要素の検索は非常に高速です。コレクション内の要素は昇順に並んでいるのですから二分検索アルゴリズムが適用できます。すなわちコレクション内の要素列の真ん中の値と検索したい値とを比較し、前者が大きければ前半分／小さければ後半分にあるでしょう。そうやって半分に絞り込まれた要素列の真ん中と比較して……、を繰り返すことで検索候補の数は1/2、1/4、1/8……と減少しますから、検索に要する時間は要素数を「N」とするとおおよそ「logN」に比例