コレクション

配列の繰り返し処理

通常、配列に対して複雑な処理を連続的に行う場合はループを使います
グラフィックスシステムの複雑な画像処理やフィルタ処理によく出てくるアルゴリズムです

このような処理には、この講座の配列のサンプルソースのように
配列の要素数を何らかの方法で取得し、それをカウンタ変数と比較して監視します

for (int i = 0 ; i < ary.Length ; i++) ...

こうすることで、配列 ary の全ての要素を繰り返し処理で操作することができますが
しかし、ループ条件の式に誤りがあれば簡単にバグにつながってしまいます
そこで、Visual Basic の **For Each...Next** ステートメントを C# でも採用したのです

VB の For Each...Next の仕様であれば、配列を安全に列挙できるのです
Microsoft は C# にもこれを採用し、要素単位で同一の処理を行う方法を提供したのです
C# では **foreach** ステートメントとして次のような構文を持ちます

foreach (type identifier in expression) statement

この構文は Perl 言語の foreach 文によく似ているでしょう
type には型、identifier には識別子、expression には配列を指定します

foreach は expression で指定した配列の全要素分だけ statement を繰り返し実行します
statement は配列を処理するための埋め込みステートメントです
実は、繰り返すごとに expression の要素が 0 番から順に identifier に格納されます
statement で配列の要素にアクセスするには、この identifier を参照すれば良いのです

foreach(string str in str_ary) GetString(str);

例えば上の例は、str_ary という文字列型配列が存在するとして
この配列の全要素を順に GetString(string) メソッドに渡して処理させたいとします
foreach は str_ary のインデックス 0 番を一時的なローカル変数 str に保存し
その後、埋め込みステートメントである GetString(str) を実行します

埋め込みステートメントのエンドポイントに到達すれば、再び foreach の先頭に復帰し
今度は str_ary のインデックス 1 番を str に保存して埋め込みステートメントを実行します
これが、str_ary.Length - 1 回繰り返されるのです

```
class Test {
 static void Main() {
  string[] str = {
   "Kitty on your lap" ,
   "Silver Gene" ,
   "Di Gi Charat" ,
   "Tokyo mew mew"
  };

  foreach(string tmp in str)
   System.Console.WriteLine(tmp);
 }
}
```

これを実行すれば、配列 str の文字列が順にコンソールに出力されます

ところで、Perl や Visual Basic とは異なり、C# は一貫したかつ正確な型情報を持ちます
C# の曖昧性のない型情報を考えれば foreach に指定できる式の型は何でしょう？

一般には foreach で指定する型は「配列である」と説明します
しかし、正確に説明するとこれは配列ではなく**コレクション型**と呼ばれるもので
System.Array もコレクション型であるという表現がもっとも適切でしょう

コレクション型の性質を網羅するには、インターフェイスについて詳しい知識が必要です
ここではまだインターフェイスについて解説していないので詳しくは説明しませんが
単純に説明すると GetEnumerator() インスタンスメソッドを持つクラスをコレクション型と言います
より詳しく知りたいならば .NET クラスライブラリの構造をよく研究すると良いでしょう

---

foreach

foreach (type identifier in expression) statement

コレクション型を評価し、埋め込みステートメントを実行します

type - 一時的な仮変数の型を指定します
identifier - 一時的な仮変数の識別子を指定します
expression - コレクション型の式を指定します
statement - 埋め込みステートメントを指定します
