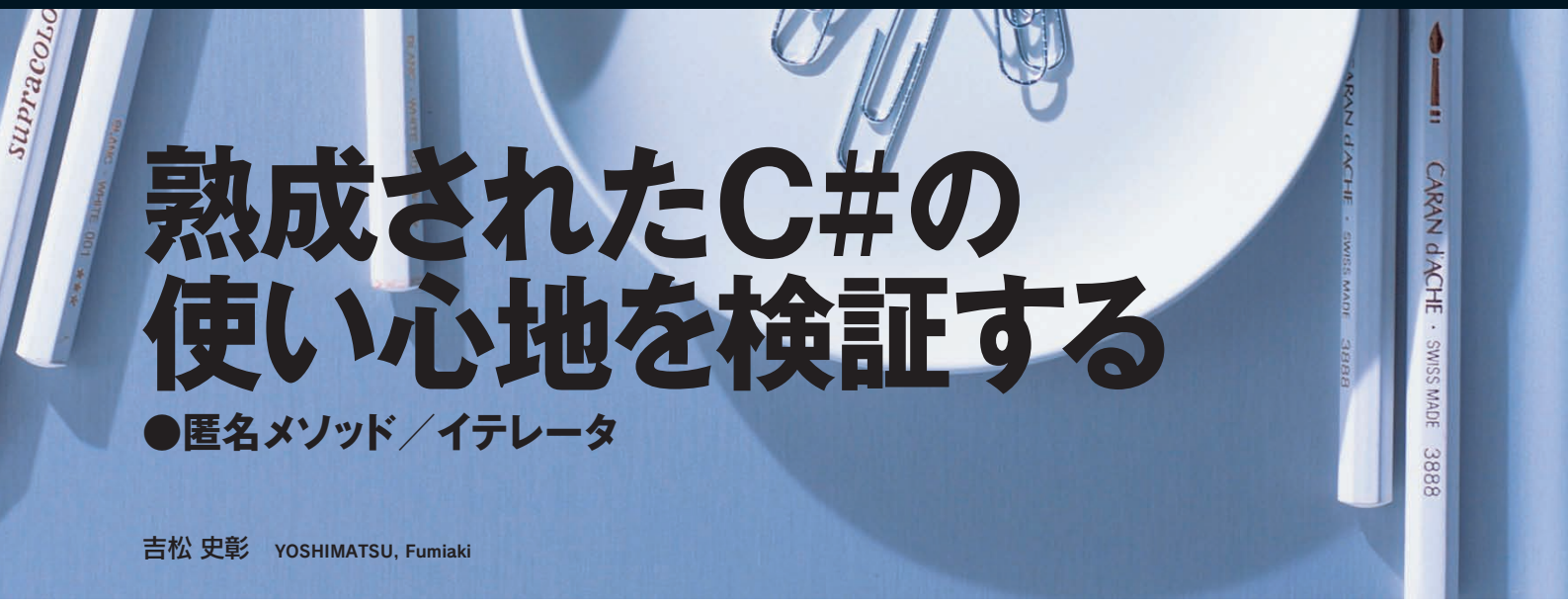

熟成されたC#の使い心地を検証する

●匿名メソッド／イテレータ

吉松 史彰　YOSHIMATSU, Fumiaki

はじめに

C#の最新バージョンには、ジェネリックなどの.NET Framework共通の新機能に加えて、C#コンパイラにも見逃せない新機能が用意されている。本稿ではその中から「匿名メソッド」と「イテレータ」を取り上げて解説する。

Level

1 2 3 4 5

Technology Tools

- ◯ Visual Basic
- ✓ Visual C#
- ◯ Visual C++
- ◯ SQL Server
- ◯ Oracle
- ◯ Access
- ◯ Excel
- ◯ ASP.NET
- ✓ Other:
  ↓
  Visual Studio 2005

匿名メソッドとは

デリゲート再訪

.NET Frameworkはオブジェクト指向に基づく型システムを重視したランタイムとクラスライブラリから構成されている。すべてのコードはメソッドとしてクラスに属しており、クラスまたはそのインスタンスであるオブジェクトを介してのみ呼び出すことができる。ただし、.NET Frameworkの得意分野でもあるデスクトップアプリケーションの開発環境では、Visual Basic以来、イベントドリブンのプログラミングが常識であり、イベントに対するハンドラを実装し、イベントが発生したときに呼び出してもらう、いわゆるコールバックの仕組みが必要になる。C言語をベースに作られているWin32 APIでは、コールバックを実現するために関数ポインタを利用する。関数ポインタは単なるポインタに過ぎず、型を意識しづらいため、型システムを壊さずにメソッドを指し示すことができるように、.NET Frameworkではデリゲートが型として採用されている。

従来のデリゲートの仕組み

.NET Framework 1.0で導入されたデリゲートは、

①デリゲート型を宣言
②宣言と同型のメソッドをクラスに実装
③①の型のインスタンスを作成して②のメソッドを登録し③のインスタンスを経由してメソッドを間接的に実行する

という仕組みになっている。①で宣言したデリゲート型は、コンパイラによって特殊なクラスに展開される。

たとえば、.NET Frameworkには「System.EventHandler」というデリゲート型が宣言されている。

```
namespace System {
(略)
  public delegate void EventHandler(
    object sender, EventArgs e);
(略)
}
```

そして、Windowsフォームのコントロールには、System.EventHandler型のClickイベントが定義されている。

```
public class Control {
（略）
 public event System.EventHandler Click
（略）
}
```

Windowsフォームを実装するときは、まずSystem.EventHandlerと同型のメソッドを実装する。同型とは、戻り値と引数リストの型が一致することを指す[注1]。

```
public void Button1_Click(
 object sender, EventArgs e) {
 /* ボタンがクリックされたときの */
 /* 処理コードを記述する       */
}
```

そして、Windowsフォームの初期化時に、デリゲート型のインスタンスにメソッドを登録する。

```
button1.Click += new
 EventHandler(Button1_Click);
 /* C# 2.0ではデリゲート型の指定は */
 /* 省略できるようになったため、  */
 /* 以下の記述でも正しく動作する  */
 // button1.Click += Button1_Click;
```

Clickイベントが発生すると、デリゲートを経由してButton1_Clickメソッドが呼び出されることになる。

匿名メソッドとは

C# 2.0で新しく採用された匿名メソッドは、名前のとおり名前のないメソッドを利用できる機能である。この機能は、従来からあったデリゲートを活用して実現している。従来のデリゲートと匿名メソッドの仕組みの違いは、

「デリゲート型のインスタンスに登録するメソッドに名前をつける必要がない」

点にある。

先のWindowsフォームの例では、Button1_Clickという名前を持ったメソッドを先に実装し、それをデリゲートに登録していた。C# 2.0の匿名メソッドを利用すると、Button1_Clickメソッドを先に実装する必要はなく、デリゲートに登録するタイミングで実装を提供できるようになる。

```
button1.Click += delegate (
 object sender, EventArgs e) {
 /* ここにボタンがクリックされたときの */
 /* 処理コードを記述する           */
};
```

言い換えると、匿名メソッドとは処理コードの集まり（コードブロック）をあたかもオブジェクトであるかのように扱える機能である。匿名メソッドのコードブロックは、デリゲート型のインスタンスであるため、匿名メソッドの主な用途は、パラメータの型としてデリゲートが指定されているメソッドに対して渡したり、デリゲートを戻り値に持つメソッドから返される場合だろう。

たとえば.NET Framework 2.0の配列（System.Arrayクラス）には、デリゲートをパラメータに取るFindメソッドが定義されている。Findメソッドは、配列の中にある特定の条件を満たす要素を探す機能を持っているが、「特定の条件を満たすかどうか」を判断するために必要なコードを、デリゲート型のパラメータとして外部から受け取り、配列の各要素を引数にしてデリゲート（に登録されているメソッド）を呼び出し、条件を満たしているかどうかを調べさせている。このメソッドは、当然従来のデリゲートを使って、クラスに「IsEven」のような名前のメソッドを定義し、それをデリゲートに登録して、デリゲートをパラメータに渡すことでも実行できる。

```
private bool IsEven(int value) {
 return (value % 2) == 0;
}

int[] α = new
 int[] { 1, 2, 3, 4, 5, 6 };
 int index = Array.Find<int>(α,
              new Predicate<int>(IsEven));
/* C# 2.0ではデリゲート型の指定は */
/* 省略できるようになったため、  */
/* 以下の記述でも正しく動作する  */
//int index = Array.Find<int>(α, IsEven);
（略）
```

しかし、匿名メソッドを利用すれば、以下のように簡潔に記述できるようになる。

```
int[] α =
 new int[] { 1, 2, 3, 4, 5, 6 };
int index = Array.Find<int>(α,
 delegate (int value) {
  return (value % 2) == 0;
 });
```

このように、匿名メソッドとは、コードブロックをあたかもオブジェクトであるかのようにパラメータとして利用できるようにする機能である。このような機能は一般にクロージャ（Closure）とも呼ばれる。クロージャは、多くのプログラミング言語に影響を与えたLISPが持っていた機能で、Ruby

注1）C# 1.xでは、戻り値と引数リストの型が完全に一致することが求められていたが、C# 2.0では型の継承関係に基づく互換性を考慮するようになっている。つまり、「一致」の条件はC# 2.0のほうがゆるいことになる。