「きれいなコードを書こう」
with C# 3.0
意図を表現編 Part II

2008/03/29

小島 富治雄
福井コンピュータ株式会社

きれいなソースコード

with C# 3.0
(VB 9.0 でも可)

きれいなソースコードについて

或る人の反論:

「ソースコードなんて汚かろうと何だろうと、
動きゃいいんだ。動きゃ」

きれいなソースコードについて

再反論 by 私

- 「動けば何でもいい」のであれば、別に汚いソースコードでなくたっていいのでは。
- きれいな場合:
  - ちゃんと動く。高品質で動く。
  - 開発コストと保守コストが少ない。
  - きれいなソースコードの方が断然得。

「美しいソースコードのための七箇条」by 私

1. **意図を表現**　← いまここ
2. 単一責務
3. 的確な名前
4. Once And Only Once
5. 的確に記述されたメソッド
6. ルールの統一
7. Testable

意図を伝えるソースコード

ソースコードは、
意図を
表現しているのが
ベター

きれいなソースコードに重要なこと

- 意図が記述されていること

かつ

- 意図以外が記述されていないこと

もし仮に…

1. ファイルを開いて、
2. その中にデータを格納し、
3. ファイルを閉じる。

というのが「意図であれば」…

ソースコードは:

```
ファイル.開く();
データ.格納(ファイル);
ファイル.閉じる();
```

の三行、が理想。

単純に…

「書きたいことを書く」

かつ

「書きたいこと以外は書かない」

が理想。

(＝ 関心事だけ分離して書く)

例えば…

もし仮に、

「10回何かする」というのが

「やりたいこと」であれば、

その意図が表現できるべき。

「10回何かする」例 *(C#1.0)*:

```
for (int i = 0; i < 10; i++)
    DoSomething();
```

int だとか 0 だとか ＋＋ だとかは、ソースコード上では「ノイズ」に過ぎない (＝意図にない)

アセンブリ言語で書いた場合の、

- ax だの 0100h のようなのと本質的には変わらない
  - 具体的なレジスタ名や番地は意図にない

C#3.0 での例:
「10回何かする」

10.回(何かする);

C#3.0を使って
意図を伝えるソースコード

C#3.0

機能がいろいろと増えて、
シンプルに意図を書くのに便利に。

- 暗黙的型付け
- パーシャルメソッド
- 自動プロパティ
- オブジェクト イニシャライザ
- コレクション イニシャライザ
- 暗黙型付け配列
- 匿名型 (Anonymous Types)
- 拡張メソッド
- ラムダ式 (Lambda Expression)
- LINQ

例：

拡張メソッド

10.回(何かする);の種あかし

```
static class 拡張
{
    public delegate void 処理();

    public static IEnumerable<int> 範囲(int ここから, int ここまで)
    {
        for (var インデックス = ここから; インデックス <= ここまで; インデックス++)
            yield return インデックス;
    }

    public static void 各々について<T>(this IEnumerable<T> コレクション, 処理 処理)
    {
        foreach (var アイテム in コレクション)
            処理();
    }

    public static void 回(this int 回数, 処理 処理)
    { 範囲(1, 回数).各々について(処理); }
}
```

拡張メソッドの例

```
static class Enumerable
{
    static IEnumerable<T>
    Where<T>(
      this IEnumerable<T> 列挙可能な何か,
      Func<T, bool> 述語
    ){ … }
    …
}
```

使用例.

```
foreach (図形 figure in 図形データ) {
   if (含まれているかどうか(描画領域, figure))
       描画(figure);
}
```

```
図形データ.
  Where  (図形 => 描画領域.含む(図形)).
  ForEach(図形 => 図形.描画                 );
```

```
図形データ.
        Where  (図形 => 描画領域.含む(図形)).
        ForEach(図形 => 図形.描画                );
```

美しい!

```
foreach (図形 figure in 図形データ) {
    if (含まれているかどうか(描画領域, figure))
        描画(figure);
}
```

関心が混在

関心が分離

```
図形データ.
  Where  (図形 => 描画領域.含む(図形)).
  ForEach(図形 => 図形.描画                    );
```

Where(条件式)

どこが美しいかというと…

- –汎用的でシンプルなアルゴリズム
- –単一責務
- –責務をメソッドに委譲
- –パラメータが一つで戻り値も一つ

→ 高凝集 *(High Cohesion)*

高凝集 (High Cohesion)

- シンプルなプログラム単位に、
- ひとつの関心事のみを、
- 記述しつくしている

「シンプルなやり方で、
必要なものだけを、
十分に書ききる」

```
static IEnumerable<T>
Where<T>(this IEnumerable<T> 列挙可能な何か,
         Func<T, bool> 述語)
```

- Where(条件式)

拡張メソッドによる
メソッド チェーン

Select(OrderBy(Where(書棚, 行抽出条件),
　並べ替え条件), 列抽出条件)

書棚.
　Where　(行抽出条件　).
　OrderBy(並べ替え条件).
　Select　(列抽出条件　);

美しい!

メソッド チェーン

```
new[] {
        new { 番号 = 1, 名前 = "伊藤博文" , 出身="山口"   },
        new { 番号 = 2, 名前 = "黒田清隆" , 出身="鹿児島" },
        new { 番号 = 3, 名前 = "三條實美" , 出身="京都"   },
        new { 番号 = 4, 名前 = "山縣有朋" , 出身="山口"   },
        new { 番号 = 5, 名前 = "松方正義" , 出身="鹿児島" },
        new { 番号 = 6, 名前 = "大隈重信" , 出身="佐賀"   },
        new { 番号 = 7, 名前 = "桂太郎"   , 出身="山口"   },
        new { 番号 = 8, 名前 = "西園寺公望", 出身="京都"   }
    }
     .Where  (首相 => 首相.出身 == "山口"          )
     .OrderBy(首相 => 首相.名前                    )
     .Select (首相 => new { 首相.名前, 首相.出身 })
     .ForEach(首相 => Console.WriteLine(首相)     );
```

型推論 (var)

型推論 (匿名型)

var 或る本 = 書棚[何冊目か];

- Haskell などではお馴染みの機能が C#3.0 に付いた

型推論 (匿名型)

```
Book b = array[n];
```

```
var 選択された本 = 本棚[ユーザーの選択];
```

「意図は型でなく、**名前**で表現すべき」

C#3.0

- 型推論 (var) をはじめとして、型の記述が不要になる方向へ進化
- でも、静的型チェックはこれまで通り
  - 動的型なし言語の場合とは違う

C#3.0 の var

- 長所:
  - 意図からすればノイズに当たる型の記述が不要!
  - 静的な検証＋動的な検証は健在!

匿名メソッドとラムダ式

```
class 掛け算器
{
    public class 積
    {
        double number;

        public double 数
        {
            get { return number; }
            set { number = value; }
        }

        public 積(double 数)
        { this.数 = 数; }
    }

    積 product;

    public 掛け算器(積 積)
    { product = 積; }

    public void アクション(double 数)
    { product.数 *= 数; }
}

static double 相乗平均(double[] データ)
{
    if (データ.Length == 0)
        throw new
        ArgumentOutOfRangeException();

    掛け算器.積 積       = new 掛け算器.積(1.0);
    掛け算器    掛け算器 = new 掛け算器   (積);
    ForEach(データ, new Action(掛け算器.アク
        ション));
    return Math.Pow(積.数, 1.0 / データ.Length);
}
```

```
static double 相乗平均(IEnumerable<double> データ)
{
    if (データ.Count() == 0)
        throw new ArgumentOutOfRangeException();

    var 積 = 1.0;
    データ.ForEach(数 => 積 *= 数);
    return Math.Pow(積, 1.0 / データ.Count());
}
```

イテレータ (yield、C#2.0)

```
class 自然数群
{
    public class Enumerator
    {
        readonly int min = 1;
        readonly int max = 1;
        int current = 0;

        public Enumerator(int min, int max)
        {
            this.min = min;
            this.max = max;
            current = min - 1;
        }

        public int Current
        { get { return current; } }

        public bool MoveNext()
        {
            if (current < max) {
                current++;
                return true;
            }
            return false;
        }
    }

    readonly int min = 1;
    readonly int max = 1;

    public 自然数群(int min, int max)
    {
        this.min = min;
        this.max = max;
    }

    public Enumerator GetEnumerator()
    { return new Enumerator(min, max); }
}
```

```
class 自然数群
{
    readonly int min = 1;
    readonly int max = 1;

    public 自然数群(int min, int max)
    {
        this.min = min;
        this.max = max;
    }

    public IEnumerator<int> GetEnumerator()
    {
        for (int n = min; n <= max; n++)
            yield return n;
    }
}
```

まとめ:

C#3.0 最強説

シンプルに意図だけを
記述できるように進化

まとめ:

C#3.0をうまく利用して
きれいなソースコードを
書きましょう

To be continued…