第22章 メソッドをもう少し詳しく見てみる

前章では、引数も、戻り値もないメソッドを作りました。この章では、引数や戻り値を持つメソッドを作ります。

作り方は簡単です。

```
戻り値の型  メソッド名 (パラメータの並び)
{
    ...// メソッドの中身
    return 戻り値;
}
```

パラメータ(parameter)は仮引数とも呼ばれます。単に引数(ひきすう)と呼ぶこともあります。これは、データ型 仮引数1, データ型 仮引数2, ...というように複数の引数を取ることも珍しくありません。この引数の値は、呼び出し側からもらっています。

最後にreturnすることで、呼び出し側に戻り値を返します。戻り値の型とメソッド名の前につけた型は一致していなくてはなりません。

```
int func(int a, int b)
{
    int c = a * b;
    return c;
}
```

とあれば、呼び出し側で引数aやbの値を指定しています。この値を使ってcの値を計算して、cを返しています。

呼び出し側では、

```
int y = func(3, 5);
```

のような感じで呼び出します。yにはfuncメソッドの戻り値が格納されます。呼び出し側の3とか5を実引数(argument)、もしくは単に引数と呼びます。引数がこのようにリテラルだとわかりやすいのですが、値を格納した変数であることもあります。

```
int a = 3, b = 5;
int y = func(a, b);
```

メソッドには、変数a,bの値を渡しているだけで変数を渡しているわけではありません。したがって、メソッドを実行後も変数a, bには何の影響もありません。これを値呼び出し(call by value)といいます。これは、非常に大事なことなので、是非理解しておいてください。

まずは、簡単なサンプルを見てみましょう。

```
// method04.cs

using System;

class Nijo
{
    public int calc(int a)
    {
        int b = a * a;
        return b;
    }
}

class method04
{
    public static void Main()
    {
        Nijo t = new Nijo();

        int x = 10;

        int z = t.calc(x);

        Console.WriteLine("{0}の2乗は{1}です", x, z);

    }
}
```

Nijoクラスは、2乗を求めるクラスです。(安直な命名法です。)

Nijoクラスのcalメソッドは、引数を2乗してこれを戻り値として返します。

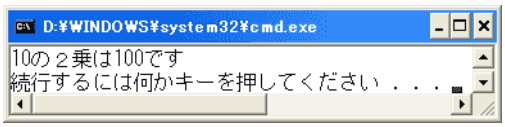
caslメソッドの中で、変数bを宣言しています。このようにメソッドの中で宣言された変数をローカル変数といいます。C/C++のローカル変数とはちょっと意味が違うので注意してください。ローカル変数は、メソッドが復帰した時点で、その値は無効となってしまいます。(次回そのメソッドが呼ばれても以前のローカル変数の値は無効)。

Mainメソッドでは、Nijoクラスのインスタンスtを生成して、calcメソッドを呼び出していいます。

```
int z = t.calc(x);
```

変数zには、calcメソッドの戻り値が格納されます。

実行結果は、次のようになります。



次に、昔からC/C++の教科書に載っている値を交換する間違ったプログラムを紹介しましょう。

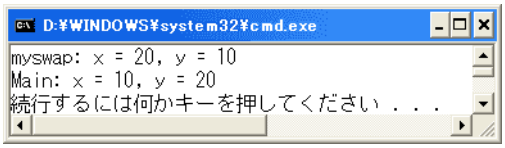
```
// swap01.cs

using System;

class MySwap
{
    public int myswap(int x, int y)
    {
        int temp;
        temp = x;
        x = y;
        y = temp;
        Console.WriteLine("myswap: x = {0}, y = {1}", x, y);
        return 0;
    }
}

class swap01
{
    public static void Main()
    {
        MySwap ms = new MySwap();

        int x = 10, y = 20;
        ms.myswap(x, y);

        Console.WriteLine("Main: x = {0}, y = {1}", x, y);
    }
}
```

myswapメソッドでは、引数xの値を一旦tempに格納しています。次に、xにyの値を代入しています。最後に、tempの値(最初のxの値)をyに代入しています。これで、xとyの値が入れ替わったはずです。念のためConsole.WriteLineメソッドでxやyの値を表示しています。

呼び出し側のMainメソッドでは、xに10, yに20を代入してms.myswap(x, y);を実行しています。

先ほども書きましたように、呼び出し側ではxやyの値をメソッドに渡しているだけで、変数そのものを渡しているわけではありません。呼び出し側のxやyの値は、何の影響も受けないはずです。

ちなみに、myswap関数は常に0を返していますが、意味はありません。



メソッド内では、x, yの値は入れ替わっていますが、呼び出し側では引数に何の影響もありません。

これでは、おもしろくありません。C/C++ならポインタを使って値を入れ替えました。

C#でも使おうと思えばポインタは使えます。では、呼び出し側で渡した引数の値を入れ替えるプログラムを作ってみましょう。

```
// swap02.cs

using System;

class MySwap
{
    unsafe public void myswap(int *x, int *y)
    {
        int temp;
        temp = *x;
        *x = *y;
        *y = temp;
    }
```

```
}


class swap02
{
    unsafe public static void Main()
    {
        int x = 10, y = 20;

        MySwap ms = new MySwap();

        ms.myswap(&x, &y);
        Console.WriteLine("x = {0}, y = {1}", x, y);

    }
}
```

呼び出し側で

```
ms.myswap(&x, &y);
```

としている点に注目してください。変数に「&」をつけると、その変数のアドレスを表します。メソッドの引数に、変数のアドレスを渡すとメソッド側で、直接変数に加工ができるようになります。

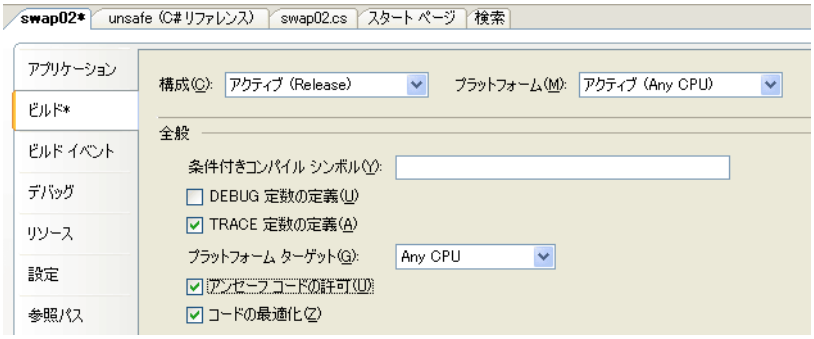
ただしC#では、常にアドレスを使えるわけではなくunsafeキーワードのついたブロック内等でしか利用できません。メモリのアドレスを扱うときは充分注意しないと、何が起るかわかりません。それで、unsafeなのかもしれません。

メソッド側の引数は

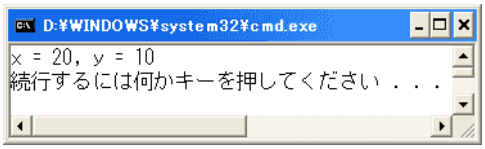
```
myswap(int *x, int *y)
```

のように、「*」がついています。これは、アドレスが示す変数の値を表します。

さらに、unsafeキーワードを使うときには、コンパイラにその旨を知らせなくてはいけません。VS2005なら、ソリューションエクスプローラでプロジェクト名を右クリックし、プロパティを表示させ、「アンセーフコードの許可」にチェックをつけなくてはなりません。



実行結果は、次のようになります。



今度は、呼び出し側のx, yの値が入れ替わっていますね。

C/C++では、今のような方法を頻繁に用いますが、C#ではあまり使いません。渡した変数の値に加工をしてもらいたい場合には「ref」キーワードを使います。これを「参照による呼び出し(call by reference)」と呼びます。

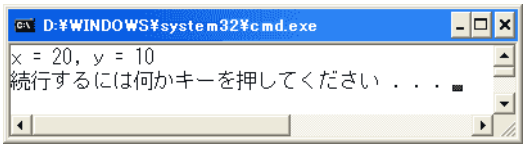
メソッド側：データ型 メソッド名(ref データ型 変数名,....){...}
呼び出し側：メソッド名(ref 変数名,...);

では、引数の値を交換するプログラムを作ってみましょう。

```
// swap03.cs

using System;

class MySwap
{
    public void myswap(ref int x, ref int y)
    {
        int temp;
        temp = x;
        x = y;
        y = temp;
    }
```

```
}

class swap03
{
    public static void Main()
    {
        int x = 10, y = 20;

        MySwap ms = new MySwap();
        ms.myswap(ref x, ref y);

        Console.WriteLine("x = {0}, y = {1}", x, y);

    }
}
```

では、実行結果を見てみましょう。



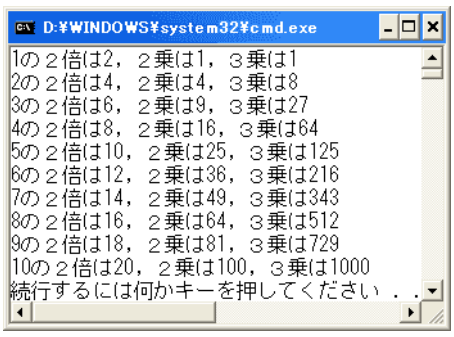
x, yの値が入れ替わっていますね。

さて、refキーワードをつけた引数(変数)は、初期化されていないといけません。(引数の値を加工してもらうので、最初から値が指定されていないと加工のしようがない!)

これに対して、outキーワードは初期化しておく必要はありません。呼び出し側でも、メソッド側でも引数の前にoutキーワードをつけておく必要があります。

```
// method05.cs

using System;

class MyOut
{
    public void calc(int x, out int i, out int j, out int k)
    {
        i = x * 2;
        j = x * x;
        k = x * x * x;
    }
}

class method05
{
    public static void Main()
    {
        MyOut mo = new MyOut();
        int a = 3, x, y, z;

        mo.calc(a, out x, out y, out z);

        for (int i = 1; i <= 10; i++)
        {
            mo.calc(i, out x, out y, out z);
            Console.WriteLine("{0}の2倍は{1}, 2乗は{2}, 3乗は{3}",
                i, x, y, z);
        }

    }
}
```

実行結果は次のようになります。



Update 28/Aug/2006 By Y.Kumei