構造体

値型のクラス

C# 言語(より正確には .NET の共通言語仕様)は、「構造体」という概念を持ちます
C/C++ 言語を習得している方は、すでに構造体は馴染み深いものでしょう
C++ 言語の構造体はデフォルトのアクセスレベルが異なるという点以外で「クラス」と同義でした

しかし、C# 言語において構造体はクラスと大きく異なる部分があります
.NET アプリケーションのクラスのインスタンスは必ずヒープに割り当てられました
構造体は、これに対してスタックに配置される、すなわち値型という形をとります

構造体を用いれば、クラスとは異なりオブジェクトを配置するオーバーヘッドはなくなります
値型であるということ以外は、基本的にクラスと同じ扱いをすることが可能です
構造体の宣言には **struct** キーワードと次の構文を用います

[attributes] [modifiers] struct identifier [:interfaces] body [;]

クラスの宣言時の class キーワードが struct キーワードに変化しただけです
構造体型の変数は参照型ではないため、実体はスタックにあることを忘れないでください

```
struct Kitty {
 public string str;
 public Kitty(string str) {
  this.str = str;
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty("Kitty on your lap");
  getKitty(obj);
  System.Console.WriteLine(obj.str);
 }

 public static void GetKitty(Kitty obj) {
  obj.str = "Tokyo mew mew";
 }
}
```

このプログラムは、構造体とクラスの違いを知ることができます
Kitty 構造体はメンバ変数 str フィールドを持っています
Kitty 構造体の変数 obj を生成した後、getKitty() メソッドにこの変数を渡しています
ただし GetKitty(obj) で渡しているのは、参照ではなく obj のコピーなのです

そのため、GetKitty() メソッド内で obj を変更しても Main() の obj 変数に変化はありません
また、構造体は値型なので非初期化状態でも null にはなりません
構造体の場合は、フィールドのスタック領域が 0 で埋められます

```
struct Kitty {
 public int i;
}

class Test {
 static Kitty obj;
 static void Main() {
  System.Console.WriteLine(obj.i);
 }
}
```

Main() メソッドで呼び出された obj は初期化されていないので
構造体のフィールドは 0 で埋められた状態です
通常、初期化されていない可能性のあるコンパイルはエラー、または警告が出ます

しつこいようですが、注意しなければならないのは「構造体は値型」ということです
クラス型変数の場合、代入は特に意識することなく行うことができました
参照型の代入は、参照先アドレスのコピーでありインスタンスをコピーすることではありません

しかし、値型変数を代入した場合は**スタック領域を丸ごとコピー**してしまいます
もし巨大なフィールドを持つ構造体を代入コピーしたり
またはメソッドに値渡しする場合、膨大なスタック領域を使用してしまうかもしれません
C 言語経験者はご存知だと思いますが、これは無駄なオーバーヘッドを生み出します

```
struct Kitty {
 public string str;
 public Kitty(string str) {
  this.str = str;
 }
}

class Test {
 static void Main() {
  Kitty obj1 = new Kitty("Kitty on your lap");
  Kitty obj2 = obj1;
  obj2.str = "Silver gene";
  System.Console.WriteLine(obj1.str);
  System.Console.WriteLine(obj2.str);
 }
```

```
}
```

このプログラムは、構造体の代入がスタックのコピーであることを表します
obj1 と obj2 に物理的な関係はないということを知ることができるでしょう
struct Kitty という文を class Kitty と変えるだけで、その違いを見ることができるはずだ

では、構造体をメソッドに渡す時はどうすればよいでしょう
値渡しをすれば無駄なオーバーヘッドが生まれる（コピーを目的としない場合）
このとき、C 言語ではポインタを、C++ では参照を関数に渡すという方法を用いました
C# も同様に、値の参照をメソッドに渡す ref や out を用いるという方法が妥当でしょう

```
struct Kitty {
 public string str;
 public Kitty(string str) {
  this.str = str;
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty("Kitty on your lap");
  GetKitty(ref obj);
  System.Console.WriteLine(obj.str);
 }
 static void GetKitty(ref Kitty obj) {
  System.Console.WriteLine(obj.str);
  obj.str = "Silver Gene";
 }
}
```

このプログラムは、GetKitty メソッドに Kitty 構造体を渡す手段として参照渡しを用います
GetKitty() に渡されるのは、実際には Kitty へのポインタなので実体はコピーされません

構造体のコンストラクタ

構造体のコンストラクタには、クラスと異なりいくつかの制約があります
まず、構造体には値を受け取らないデフォルトコンストラクタを指定できません
デフォルトコンストラクタは最初から定義されていて、フィールドを 0 で初期化します

このため、構造体のフィールドはデフォルトで常に 0 です
構造体のインスタンスフィールドに**初期化子は許可されません**
構造体の初期化子は、静的フィールドにのみ指定することができます
さらに構造体のコンストラクタに base(...) を指定することはできません

```
struct Kitty {
 public static string s_str = "Silver Gene";
 public string str;  // = "Kitty on your lap"; //とするとエラー
 //public Kitty(); //ノンパラメータのコンストラクタは作れない
 public Kitty(string str) {
  //base(); //base() コンストラクタ初期化子は使えない
  this.str = str;
 }
}

class Test {
 static void Main() {
  Kitty obj = new Kitty("Kitty on your lap");
  System.Console.WriteLine(Kitty.s_str);
  System.Console.WriteLine(obj.str);
 }
}
```

これらの構造体の規制をクラスと混同させないでほしい
もっとも、通常のプログラムでは独自の構造体を作る必要はそれほどない

---

this 変数

クラスのインスタンスメソッドにおける this は自己のインスタンスへの参照でした
しかし、構造体の this はインスタンスの参照という点では同じですが、その性質が違います

クラスのインスタンスメソッドにおける this は**リテラル**に分類されます
つまりこれは値であり、変数とは異なる「定数」であるという扱いです
しかし、構造体のインスタンスメソッドにおける this は
自己のインスタンスの参照をデフォルトとする、自己構造体型の**変数**である

リテラルである this は読み込み専門であり代入はできません
ところが変数である構造体の this には、その構造体を代入することができるのです
正確には、コンストラクタにおける this は構造体型の out パラメータであり
インスタンスメソッドにおける this は ref パラメータに相当すると考えることができます

```
struct Kitty {
 public string str;
 public Kitty(Kitty obj) {
  this = obj;
  str = this.str;
 }
 public Kitty(string str) {
  this.str = str;
 }
```

```
}

class Test {
 static void Main() {
  Kitty tmp = new Kitty("Kitty on your lap");
  Kitty obj = new Kitty(tmp);
  System.Console.WriteLine(obj.str);
 }
}
```

このプログラムの処理内容については、意味のあるものではありません
重要なのは this = obj **という見なれない** this **への代入文です**
this **が変数であるといって、通常は無意味に** this **に代入するようなことはしません**
しかし、構造体のインスタンスメソッドにおける this **が変数であるということを知ることは重要です**

前のページへ　戻る　次のページへ