C#

От основ к эффективному коду

Дейнега Василий Михайлович

It Works (itw66.ru)

Net Framework

История:

- Java..
- Одна платформа на множестве устройств

Состав:

- 1. Common Language Runtime (CLR)
- 2. Framework Class Library (FCL)

Компиляция:

- Code -> Common Intermediate Language (IL) -> Processor Commands
- Множество языков (C#, C++, J#, VB..) + Множество платформ (x86, a64, Alpha, PowerPC..)
- Высокая скорость

Простота:

- Сборка мусора
- Визуальный редактор форм
- Проверка безопасности типов
- Простота развертки приложений, нет dll hell, есть GAC. Метаданные и код в одной сборке

Базовые типы

Все типы производные от System.Object!

- Int8, Int16, Int32, Int64, UInt8…
- Single, Double
- bool (Boolean)
- Char, byte
- String или string???
- Object

Функции:

- Не может быть в неймспейсе
- Не может иметь параметров по умолчанию
- Нет friend функций

using:

- using Targem.Controls;
- using ImageList = List< Image>; // аналог typedef

class/struct

Reference type (ссылочный тип, class):

- Располагаются в куче и передается по ссылке (умный указатель в C++)
- new выделяет для них память
- Имеют дополнительные поля

Value type (размерный тип, struct, enum):

- Значительно эффективнее, не имеют дополнительных полей, не делается разыменование
- Располагается в разных местах (чаще в стеке), но передается по значению
- new не выделяет память, а инициализирует структуру
- Не обрабатываются сборщиком мусора

Упаковка – преобразование **val -> ref:**

```
Int32 val = 10;
Object obj = val;
val = (Int32)obj;
```

Дорогая операция!!!

class

Функции класса Object:

- Type GetType()
- string ToString()
- int GetHashCode()
- bool Equals(object obj)
- Finalize

Видимость класса:

- public, internal, private, sealed

Видимость типов, полей, методов:

- public, internal, protected, private

Аттрибуты методов:

- static, virtual, override, abstract

Некий класс:

```
public class SomeClass
{
        private Int32       m_value = 0;
        public Int32        Value { get; set; }

        public SomeClass(){}
}
```

Сведения:

- Нет множественного наследования, есть много интерфейсов
- Равенство, тождество

struct

Важно:

- Массив структур значительно эффективнее массива классов

```
class IntClass      {   public Int32 value;   }
struct IntStruct    {   public Int32 value;   }
```

  - Создание массива быстрее в 86 раз
  - Проход по массиву с записей в переменную быстрее в 5 раз
- Если при работе с структурами придется преобразовывать к Object, то лучше делать class (ArrayList).

Enum:

1 Вариант:

```
public enum TestEnum{ One = 0, Two, Tree }
TestEnum  t = TestEnum.One;
Int32 it = (Int32)t;
```

2 Вариант:

```
public enum TestEnum : Int32 { One = 0, Two, Tree }
```

3 Вариант:

```
[ Flags ]
public enum TestEnum{ One = 0x01, Two = 0x02, Tree  = 0x04}
```

Функции, свойства, аттрибуты

Функции:

```
void        Func( ref Int32 v1, out Int32 v2 )    {    v1 = v2 = 10;    }
Int32 v1 = 0, v2 = 0;
Func( ref v1, out v2 );
```

Свойства:

1 Вариант:

```
private Int32       m_value;
public Int32        Value{    get{ return m_value; }       set{ m_value = value; }      }
```

2 Вариант:

```
public Int32        Value{ get; set; }
```

3 Вариант:

```
private Int32[]      m_arr ;
public Int32        this[ Int32 index ] // свойство с параметром
{
        get{ return m_arr[ index ]; }
        set{ m_arr[ index ]= value; }
}
obj[ 10 ] = 100;
```

Аттрибуты:

```
[
   Category( “Object" ),
   DisplayName( “Some Value" ),
   Description( “This is description of value" ),
]
 public Int32          Value{ get; set; }
```

Базовые операции

Операции:

- */% ||&&<>= - базовые операции как в C++
- return obj1 ?? Obj2; // return obj1!=null ? obj1 : obj2;

Приведение типов:

- Явное приведение типа генерирует исключение при ошибке
- is – проверка класса на принадлижность типу
- as – безопасное приведение к типу. Вернет null, если привести не удалось
- MyStruct str = (MyStruct)obj; // приведение типов к структуре или Enum

Предпочтительнее так:

```
SomeClass sc = obj as SomeClass;
if( null != sc )
{
 /* do something */
}
```

Чем так:

```
if( obj is SomeClass )
{
      SomeClass sc = obj as SomeClass;
      /* do something */
}
```

Обработка исключений

Обработка исключений:

```
try
{
}
catch ( Exception exc )
{
      Dbg.LogError( exc.ToString( ) );
}
finally
{
}
```

IDisposable:

```
private class Starter : IDisposable
{
      public Starter( Object obj ){     Start();     }
      public void Dispose()        {     Stop();     }
}
```

Использование:

```
void DoSomething()
{
      using( new Starter() )
      {
      }
}
```

Работа со строками

String:

- Split, IndexOf, Trim, Remove, Replace…
- If(str. Equals(“name”) || str == “name”) // сравнение
- String str = @”hello\no”; // все символы строки используются как есть

- String res = “name: “ + strName + “ age: “ + age; // правильно, но очень медленно
- String.Format(“{0} {1}”, strName, age); // в 2 раза быстрее предыдущего примера

- Convert.ToString(intValue); // эффективнее чем intValue.ToString();
- if(String.IsNullOrEmpty(name)) // проверка на null и нулевой строки

StringBuilder: - для формирования длинны строк

- StringBuilder sb = new StringBuilder();
- sb.Append(..); // добавление элемента в конец
- sb.AppendLine(..); // добавление элемента в конец + перевод строки

Предпочтительнее так (2.35s):

```
sb.Append( "name: " );
sb.Append( strName );
sb.Append( "age: " );
sb.Append( age);
```

Чем так (3.2s + 35%):

```
sb.Append( "name: " + strName + " age: " + age );
```

Делегаты и лямбда функции

Делегат и лямбда функция:

1. ```
   delegate int Del( int v );
   void RegisterCallback(Del del ); // передача делегата
   ```
2. ```
   Del del = x => x *x; // создание делегата через лямбда функцию
   ```

Способы передачи функций:

1. ```
   void ActionFunc()
   {
       // do something
   }
   obj.RegisterCallback( ActionFunc ); // функция переданная делегатом
   ```

2. ```
   obj.RegisterCallback( delegate() // анонимный делегат, в 3 раза быстрее предыдущего
     {
           // do something
     } );
   ```

3. ```
   obj.RegisterCallback( () => { /*do something*/} ); // лямбда функция Одинаково с предыдущим
   ```
   ```

Контейнеры

Массивы:

- Int32[] arr = new Int32[count];
- Int32[] arr = new Int32[] { 1, 2, 3, 4, 5 };
- Int32[] arr = { 1, 2, 3, 4, 5};
- Int32[,] arr = new Int32[2, 3];
- Int32[,] arr = { { 1, 2, 3 }, { 4, 5, 6 } }; // массивы в виде матриц
- Int32[][] arr = new Int32[6][]; // массивы разной размерности для каждой строки

- arr – это объект класса Array
- Array.Resize, Copy, Sort, Find

- Массивы структур значительно эффективнее массивов классов

Контейнеры

List< Int32 >:

```
List< Int32 > arr = new List< Int32 >( count );
Int32 val = arr[ 10 ];
arr.Add(); Remove, Clear, Sort, IndexOf
```

ArrayList ⇔ List< Object >

Dictionary< key, value > (Hashtable):

```
Dictionary< String, SomeClass > arr = new Dictionary< String, SomeClass >
arr[ “name” ] = “newValue”; // установка значения
String val = arr[ “name” ]; // получение значения
```

Предпочтительнее так (7.64s):

```
Image img = null;
if( images.TryGetValue( name, out img ) )
    return img;
```

Чем так (11.29s + 47%):

```
if( images.ContainsKey( name ) )
    return images[ name ];
```

Контейнеры

Компактная обработка массива:

1. list.ForEach(item => item.DoSomething());
2. SomeClass item = list.Find(item => item.Name == name);
3. List<SomeClass> items = list.FindAll(item => item.Value > 10);
4. bool res = list.TrueForAll(item => item.IsValid);

5. foreach(var pair in dict) лучше чем так

   foreach(KeyValuePair< String, SomeClass > pair in dict)

Нужные компоненты

Math:

- Math.Abs, Min, Max, Sin, Cos…

Файлы:

- System.Xml: XmlDocument, XmlNode, XmlReader, XmlWriter…
- System.IO: File, Path, Directory, StreamReader, StreamWriter, FileStream, MemoryStream…

Элементы GUI:

- System.Windows.Forms
- System.Drawing (GDI, GDI+)

- Reflection (RTTI)
- Linq
- Actions
- …

Рекомендуемая литература

- Программирование на платформе Microsoft .NET Framework, Рихтер Дж. [Русская редакция, 2003]
- Professional C#, Simon Robinson [Wrox Press, 2001]
- http://rsdn.ru – полезные статьи на русском
- http://www.codeproject.com/ - много примеров реализации, очень помогает для создания своих визуальных компонентов

Спасибо за внимание