Структуры и перечисления

Структуры

Понятие структуры.

Синтаксис объявления структуры.

Необходимость и особенности применения структур.

Понятие структуры.

Структура - это конструкция языка, позволяющая содержать в себе набор полей различных типов.

Синтаксис объявления структуры.

```
struct имя : интерфейсы
{
  // объявления членов
}
```

```
struct Struct
{
    int field;
    static int second;
}
```

В C#.NET **структуры** предназначены для группирования и хранения небольших групп связанных данных.

Отличие структур в C#

1. Структуры – это **типы значений**, а не ссылочные типы.
2. Структуры сохраняются в стеке, либо в куче (**если являются полями объекта класса**).
3. Структуры имеют ограничения времени жизни, что и **простые типы данных**.
4. Структуры могут содержать **конструкторы, константы, поля, методы, свойства, индексаторы, операторы, события и вложенные типы**.
5. В структурах нельзя инициализировать поля **непосредственно в месте создания**. Инициализация статических полей необязательна.
6. В структурах допускается также определять конструкторы, но не деструкторы. **Важно! Конструктор, используемый по умолчанию не подлежит изменению.**

Особенности использования структур

- структуры **не могут наследовать другие структуры и классы**
- структуры **не могут выступать в качестве базовых** для других структур и классов
- **наследуются «неявно»** от абстрактного класса **ValueType**, который наследуется от **класса Object**
- в структуре **можно реализовать** один или несколько **интерфейсов**.
- т.к. структуры не поддерживают наследование, то их члены нельзя указывать как **abstract**, **virtual** или **protected**.

Пример структуры.

```
struct Book  // объявление структуры
{
 public int Id;  // !!!!
 public string Name;
 public int CountPages=100; // Ошибка – нельзя инициализировать

 // ОШИБКА!
 // Структуры не могут явно содержать конструкторы без параметров
 // т.к. он реализуется автоматически
 //public Book()
 //{   Id=1;     //}

//  конструктор с параметрами
// здесь должны быть проинициализированы все поля
 public Book(int I, string N, int CP)
 {    Id = I; Name = N; CountPages = CP;
 }

 public override string ToString()
 {
   return String.Format("№{0}. {1}, стр. {2}", Id, Name, CountPages);
 }
}
```

```
// создание структуры с инициализацией ч/з конструктор
Book b1 = new Book(1, "Отцы и дети", 250);
Console.WriteLine(b1);

// создание структуры без инициализации
Book b2;
// здесь должны быть проинициализированы все поля
// т.к. обращение к непроинициализированной структуре
b2.Id = 2; b2.Name = "Война и мир";  b2.CountPages = 500;
Console.WriteLine(b2);

// создание структуры с инициализацией ч/з конструктор по умолчанию
Book b3 = new Book();
b3.Id = 3; b3.Name = "Му-му"; b3.CountPages = 300;
Console.WriteLine(b3);

// копирование структур (создание двух копий)
Book b4 = b3;
b3.Id = 4; b3.Name = "Обломов"; b3.CountPages = 190;

Console.WriteLine(b3);
Console.WriteLine(b4);
b5 = b4;
Console.WriteLine(b5);
```

```
// Структуры.
// В структурах можно создавать автоматически реализуемые свойства,
// при этом требуется использовать конструктор при построении
экземпляра.

   struct MyStruct
    {
        public int MyProperty { get; set; }
    }

    class Program
    {
        static void Main()
        {
            MyStruct instance = new MyStruct();  // ОБЯЗАТЕЛЬНО

            instance.MyProperty = 1;
            Console.WriteLine(instance.MyProperty);

            Console.ReadKey();
        }
    }
```

```
// Структуры могут содержать статические члены.

struct MyStruct   // Статические структуры недопустимы.
 {
   public static int Field
    {    get;   set;    }
   // Статический конструктор всегда отрабатывает первым.
   static MyStruct()
    {   Console.WriteLine("Static Constructor");       }

   public static void Show()
     {    Console.WriteLine(Field);      }
}

 class Program
{
        static void Main()
        {
            // Инициализация статических полей необязательна.
            //MyStruct.Field = 1;
            MyStruct.Show();
         }
}
```

Необходимость и особенности применения структур

Структуры используют для **оптимизации производительности**, когда нужно только хранение некоторых небольших порций данных. Оптимизация производительности достигается за счет того, что структура **хранится в стеке** и доступ к ней осуществляется напрямую.

При этом…

При **присваивании одной структуры другой или при передаче методу структуры**, как параметра, происходит **полное копирование** содержимого структуры, что **намного медленнее передачи ссылки.**

- Структура **DateTime** представляет текущее время, обычно выраженное как **дата и время суток**.
- Тип значения **DateTime** представляет дату и время в диапазоне от 00:00:00 1 января 0001 года (н. э.) до 23:59:59 31 декабря 9999 года (н. э.)

```
// DateTime.Now возвращает объект System.DateTime,
// которому присвоены текущие дата и время суток данного компьютера
DateTime now = DateTime.Now;

Console.WriteLine("Текущая дата и время : {0}", now);
Console.WriteLine("Год: {0}", now.Year);
Console.WriteLine("Месяц: {0}", now.Month);
Console.WriteLine("День месяца: {0}", now.Day);

Console.WriteLine("Текущее время - {0}:{1}:{2}",
                                  now.Hour, now.Minute, now.Second);

Console.WriteLine("День недели : {0}", now.DayOfWeek);
Console.WriteLine("Это {0}-й день года", now.DayOfYear);
```

Настройка формата времени и даты

Описатель	Описание
d	Представляет день месяца от 1 до 31. Одноразрядные числа используются без нуля в начале
dd	Представляет день месяца от 1 до 31. К одноразрядным числам в начале добавляется ноль
ddd	Сокращенное название дня недели
dddd	Полное название дня недели
f / fffffff	Представляет миллисекунды. Количество символов f указывает на число разрядов в миллисекундах
g	Представляет период или эру (например, "н. э.")
h	Часы в виде от 1 до 12. Часы с одной цифрой не дополняются нулем
hh	Часы в виде от 01 до 12. Часы с одной цифрой дополняются нулем
H	Часы в виде от 0 до 23. Часы с одной цифрой не дополняются нулем
HH	Часы в виде от 0 до 23. Часы с одной цифрой дополняются нулем
K	Часовой пояс
m	Минуты от 0 до 59. Минуты с одной цифрой не дополняются начальным нулем
mm	Минуты от 0 до 59. Минуты с одной цифрой дополняются начальным нулем
M	Месяц в виде от 1 до 12
MM	Месяц в виде от 1 до 12. Месяц с одной цифрой дополняется начальным нулем
MMM	Сокращенное название месяца
MMMM	Полное название месяца
s	Секунды в виде числа от 0 до 59. Секунды с одной цифрой не дополняются начальным нулем
ss	Секунды в виде числа от 0 до 59. Секунды с одной цифрой дополняются начальным нулем
t	Первые символы в обозначениях AM и PM
tt	AM или PM
y	Представляет год как число из одной или двух цифр. Если год имеет более двух цифр, то в результате отображаются только две младшие цифры
yy	Представляет год как число из одной или двух цифр. Если год имеет более двух цифр, то в результате отображаются только две младшие цифры. Если год имеет одну цифру, то он дополняется начальным нулем
yyy	Год из трех цифр
yyyy	Год из четырех цифр
yyyyy	Год из пяти цифр. Если в году меньше пяти цифр, то он дополняется начальными нулями
z	Представляет смещщение в часах относительно времени UTC
zz	Представляет смещение в часах относительно времени UTC. Если смещение представляет одну цифру, то она дополняется начальным нулем.

Настройка формата времени и даты

```
DateTime now = DateTime.Now;
Console.WriteLine(now.ToString("hh:mm:ss"));
Console.WriteLine(now.ToString("dd.MM.yyyy"));
```

```
05:04:43 1
7.07.2015
```

- Структура **TimeSpan** представляет интервал времени.

```
TimeSpan interval = new TimeSpan(1, 15, 42, 45, 750);
Console.WriteLine("Интервал: {0}", interval);

Console.WriteLine("Всего {0:N5} миллисекунд :",
                                        interval.TotalMilliseconds);

long nMilliseconds =    interval.Days * 24 * 60 * 60 * 1000 +
                        interval.Hours * 60 * 60 * 1000 +
                        interval.Minutes * 60 * 60 * 1000 +
                        interval.Seconds * 1000 +
                        interval.Milliseconds;
Console.WriteLine("   Миллисекунд:     {0,18:N0}", nMilliseconds);
```

```
Интервал: 1.15:42:45.7500000
Всего 142 965 750,00000 миллисекунд :
   Миллисекунд:                 142 965 750
```

Перечисления (enum)

Понятие перечисления.

Синтаксис объявления перечисления.

Необходимость и особенности применения перечисления.

Установка базового типа перечисления.

Использование методов для перечислений.

Перечисление представляет собой множество именованных целочисленных констант.

Синтаксис объявления перечисления

```
enum EnumName
{
  elem1,
  elem2,
  elem3,
  elem4
}
```

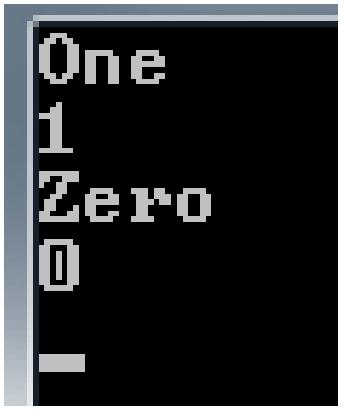
```
enum DayOfWeek
{
    Monday, Tuesday, Wednesday,
    Thursday, Friday, Saturday,
    Sunday
}
```

```
enum EnumType  // По умолчанию тип: int.
    {          // Можно явно указать: byte, sbyte,
               // short, ushort, int, uint, long, ulong
        Zero,
        One,
        Two,
        Three
    }

    class Program
    {
        static void Main()
        {
            Console.WriteLine(EnumType.One);
            Console.WriteLine((int)EnumType.One);

            EnumType digit = EnumType.Zero;
            Console.WriteLine(digit);
            Console.WriteLine((int)digit);

            // Delay.
            Console.ReadKey();
        }
    }
```



Особенности использования перечислений

- каждая символически обозначаемая константа в **перечислении имеет целое значение**
- в перечислении константе задается целое значение, которое на единицу больше, чем у предыдущей константы.
- По умолчанию **значение первой** символически обозначаемой константы в перечислении **равно нулю**.

Инициализация перечисления

```
// объявление перечисления и указание базового типа (byte)
enum Publish
{  // присваивание значений константам
   Williams,      // 0
   Jonson =11,    // 11
   Zorka,         // 12
   UN_company     // 13
}
```

```
// объявление перечисления и указание базового типа (byte)
enum Publish
{  // присваивание значений константам
   Williams = 25,  // 25
   Jonson = 10,    // 10
   Zorka = 141,    // 141
   UN_company = 2  // 2
}
```

Указание базового типа перечисления

По умолчанию в качестве базового для перечислений используется тип int, тем не менее перечисление может быть создано любого целочисленного типа, за исключением char.

```
// объявление перечисления и указание базового типа (byte)
enum Publish : byte
{
    Williams = 25,
    Jonson = 10,
    Zorka = 141,
    BHV_Piter = 2
}
```

```
enum DistanceSun : ulong
{
    Sun = 0,
    Mercury = 57900000,
    Venus = 108200000,
    Earth = 149600000,
    Mars = 227900000,
    Jupiter = 7783000000,
    Saturn = 1427000000,
    Uranus = 2870000000,
    Neptune = 4496000000,
    Pluto = 5946000000
}
```

```
// вывод имени именованной константы перечисления
Console.WriteLine(Publish.Williams);

// вывод значения именованной константы перечисления
Console.WriteLine((byte)Publish.Williams);
```

```
Style b2= Style.art;
if (b2.style == Style.fairy_tales)
{
    // ...
}
```

```
for (Style i = Style.art; i < Style.lyrics; i++)
 {
   // ...
 }
```

```
 Book b2;
 b2.Id = 2; b2.Name = "Руслан и Людмила";
 b2.CountPages = 500;
 b2.style = Style.fairy_tales;        // использование перечисления
 Console.WriteLine(b2);
switch (b2.style)
 {
     case Style.art:          Console.WriteLine(b2.style); break;
     case Style.documentary: Console.WriteLine(b2.style); break;
     case Style.fairy_tales: Console.WriteLine(b2.style); break;
     case Style.lyrics:       Console.WriteLine(b2.style); break;
     default: Console.WriteLine("не найдено!"); break;
 }
```

Использование методов для перечислений

все перечисления неявно наследуют от класса **System.Enum**, который наследует от класса **System.ValueType**, а тот — от класса **object**.

https://msdn.microsoft.com/en-us/library/System.Enum_methods(v=vs.110).aspx

Использование методов для перечислений

```
// Получение списка имен констант в перечислении
string[] names= Enum.GetNames(typeof(States));
Array arrayValues = Enum.GetValues(typeof(States));

for (int i = 0; i < names.Length; i++)
{
    Console.WriteLine(names[i] + "=" + (int)arrayValues.GetValue(i));
}

string optionsString = "Stop";
// Получение значения из строки
var result = (States)Enum.Parse(typeof(States), optionsString);
Console.WriteLine("Значение Stop: "+ (int)result);
```

```
enum States
    {
        Play    =10,
        Stop    =12,
        Pause   =13,
        Rec     =14
    }
```

```
Play=10
Stop=12
Pause=13
Rec=14
Значение Stop: 12
```

Использование перечислений (флаги)

```
[Flags]   // Использование перечисления как набор флагов
public enum CarParams
{
  HasRadio= 1,
  HasAutoPilot=2,
  HasCarpets=4
}
```

```
public class Car
   {
       public string Name { get; set; }
       public CarParams Params { get; set; }
   }
```

```
Car[] cars = new Car[2];
cars[0] = new Car()
{    Name = "BMV",
     // Добавление набора параметров
     Params = CarParams.HasRadio | CarParams.HasAutoPilot
};

cars[1] = new Car()
{  Name = "Ford",
     Params = CarParams.HasAutoPilot | CarParams.HasCarpets
};
     foreach (var car in cars)
     {
         // Проверка на наличие параметра
       if ((car.Params & CarParams.HasRadio) == CarParams.HasRadio)
       {  Console.WriteLine("У машины есть радио!");
       }
       // Проверка на наличие параметра с применением метода HasFlag
       if (car.Params.HasFlag(CarParams.HasCarpets))
       {  Console.WriteLine("В машине есть коврики!");
       }
     }
```

Необходимость применения перечислений

Использование перечислений позволяет сделать исходные коды программ более читаемыми, так как позволяют заменить «**магические числа**», кодирующие определённые значения, на читаемые имена.