C# 10 Record structs

Шипунов Илья

ГК Монополия

ishipunov@gmail.com

Mutable models

```
public class PersonName
{
    public string
        FirstName { get; set; }
    public string
        LastName { get; set; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
```

- ✓ Объявление
- ✓ Инициализация
- ✗ Контроль использования
- ✗ Трудно уловимые ошибки

Immutable models

```
public class PersonName
{
    public PersonName(
        string firstName,
        string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public string FirstName { get; }
    public string LastName { get; }
}
```

- ✓ Надежность
- ✓ Многопоточность
- ✗ Шаблонный код
- ✗ Ошибки в конструкторе
- ✗ Стоимость поддержки
- ✗ Нет неразрушающего изменения

Readonly structs (C# 7.2)

```
public readonly struct PersonName
{
    public PersonName(
        string firstName,
        string lastName)
    {
        FirstName = firstName;
        LastName = lastName;
    }

    public string FirstName { get; }
    public string LastName { get; }
}
```

- ✓ GC
- ✓ Immutable
- ✗ Шаблонный код
- ✗ Подводные камни

Record classes (C# 9.0)

```
public record PersonName(
   string FirstName, string LastName);
```

- ✓ Reference type
- ✓ Минимализм
- ✓ Структурное равенство
- ✓ Неразрушающее изменение
- ✓ ToString()
- ✗ Value type

Личный опыт

```
private static NameDto[] items =
   { new() { Name = "Sparrow" },
     new() { Name = "Turner" } };

public static
   IEnumerable<NameDto> Sort()
{
    return items.Select(x =>
       {
          x.Name = new
                string(x.Name.Reverse()
                        .ToArray());
          return x;
       })
       .OrderBy(x => x.Name);
}
```

- ✗ IEnumerable<T> + mutable DTO
- ✓ IReadOnlyXXX<T>
- ✓ Immutable DTO
- ✓ Маленькие PR

Чтобы
дважды
не
наступали ...

Record structs (C# 10.0)

```
public record struct PersonName
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
name.LastName = "Mr. Smith";
```

Readonly record structs

```
public readonly record struct PersonName
{
    public string? FirstName { get; init; }
    public string? LastName { get; init; }
}

var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
//name.LastName = "Mr. Smith";
```

Record structs

- ✓ Value type
- ✓ GC
- ✓ Минимализм
- ✓ Структурное равенство
- ✓ .NET Standard 2.0
- ✓ Реализация интерфейсов
- ✓ Generics
- ✓ Неразрушающее изменение
- ✓ Pattern matching
- ✓ ToString()
- ✘ Изменяемые по умолчанию

Positional record structs

```
public record struct PersonName(
  string FirstName,
  string LastName);

var name = new PersonName("Jack", "Sparrow");
name.LastName = "Mr. Smith";

WriteLine($"{name.FirstName} {name.LastName}");
//Jack Mr. Smith
```

Readonly positional record structs

```
public readonly record struct PersonName(
  string FirstName,
  string LastName);

var name = new PersonName("Jack", "Sparrow");
//name.LastName = "Mr. Smith";

WriteLine($"{name.FirstName} {name.LastName}");
//Jack Sparrow
```

Конструктор по умолчанию

```
var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
//name.LastName = "Mr. Smith";
```

Positional record structs

- ✓ Плюсы Record structs
- ✓ Минимализм
- ✓ Primary constructor
- ✓ Deconstruct

Переопределение .ctor (C# 10.0)

```
public readonly record struct Nickname
{
    public string? Nick { get; init; } = "Pirate";
}

var nick = new Nickname();

nick.Nick.Should().Be("Pirate");
```

Default value expression init

```
public readonly record struct Nickname
{
    public string? Nick { get; init; } = "Pirate";
}

var nick = default(Nickname);
var nickArray = new Nickname[1];

nick.Nick.Should().BeNull();
nickArray[0].Nick.Should().BeNull();
```

Реализация IEquatable<T>

```
public readonly record struct PersonName
{
    public string? FirstName { get; init; }
    public string? LastName { get; init; }
}

public struct PersonNameStruct
{
    public string? FirstName { get; set; }
    public string? LastName { get; set; }
}

typeof(PersonName).Should()
   .Implement<IEquatable<PersonName>>();
typeof(PersonNameStruct).Should()
   .NotImplement<IEquatable<PersonNameStruct>>();
```

- ✓ Equals(object? obj)
- ✓ System.IEquatable<T>
- ✓ Equals(T other)
- ✓ GetHashCode()
- ✓ operator==(T r1, T r2)
- ✓ operator!=(T r1, T r2)

Benchmarks of Equals

```
var struct1 = new PersonNameStruct
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
var struct2 = struct1;

var record1 = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
var record2 = record1;

struct1.Equals(struct2);
record1.Equals(record2);
```

```
struct1.Equals(object)
  / record1.Equals(T)
  = 25.43
```

```
| Method |       Mean |      Error |     StdDev | Ratio | RatioSD |
|------- |-----------:|-----------:|-----------:|------:|--------:|
| Struct | 220.147 ns | 1.3450 ns | 1.1923 ns | 25.43 |    0.31 |
| Record |   8.662 ns | 0.1184 ns | 0.0924 ns |  1.00 |    0.00 |
```

Benchmarks of GetHashCode

```
var struct1 = new PersonNameStruct
{
    FirstName = "Jack",
    LastName = "Sparrow",
};

var record1 = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};

struct1.GetHashCode();
record1.GetHashCode();
```

```
struct1.GetHashCode()
  / record1.GetHashCode()
  = 4.59
```

```
| Method |      Mean |     Error |    StdDev | Ratio | RatioSD |
|------- |----------:|----------:|----------:|------:|--------:|
| Struct | 80.08 ns | 0.118 ns | 0.105 ns |  4.59 |    0.02 |
| Record | 17.44 ns | 0.090 ns | 0.084 ns |  1.00 |    0.00 |
```

Реализация Equals

- EqualityComparer<T> .Default
- Equals(T other)
- Equals(object? obj)

```
public readonly record struct NicknameList
{
    public string[]? Nicknames { get; init; }
}

var first = new NicknameList
{
    Nicknames = new []{"Pirate"},
};
var second = new NicknameList
{
    Nicknames = new []{"Pirate"},
};

first.Equals(second).Should().BeFalse();
```

Pattern matching

```
public static string GetFullName(PersonName name)
{
    return name switch
    {
        { FirstName: "Jack", LastName: "Sparrow" }
            => "Captain Jack Sparrow",
        { FirstName: var firstName, LastName: var lastName }
            => $"{firstName} {lastName}",
    };
}
```

Неразрушающее изменение

```
var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};
var anotherName = name with { FirstName = "Captain Jack" };

WriteLine($"{name.FirstName} {name.LastName}");
WriteLine($"{anotherName.FirstName} {anotherName.LastName}");
//Jack Sparrow
//Captain Jack Sparrow
```

Неглубокое копирование

```
public readonly record
   struct FamousPirateName
{
    public string? FirstName { get; init; }
    public string? LastName { get; init; }

    public List<string>
   Nicknames { get; init; } = new();
}
```

```
var son = new FamousPirateName
{
    FirstName = "William",
    LastName = "Turner",
};

var father = son with { };
father.Nicknames.Add("Bootstrap Bill");

son.Nicknames.Should().HaveCount(1);
father.Nicknames.Should().HaveCount(1);
```

ToString()

```
var name = new PersonName
{
    FirstName = "Jack",
    LastName = "Sparrow",
};

WriteLine(name);
//PersonName { FirstName =
  Jack, LastName = Sparrow }
```

- ✓ Перегрузка ToString()
- ✓ Перегрузка PrintMembers()

Deconstruct

```
public readonly record struct PersonName(
  string FirstName, string LastName);

var name = new PersonName("Jack", "Sparrow");
var (firstName, lastName) = name;

WriteLine($"{firstName} {lastName}");
//Jack Sparrow
```

Pattern matching

```
public static string GetFullName(PersonName name)
{
    return name switch
    {
        ("Jack", "Sparrow") => "Captain Jack Sparrow",
        var (firstName, lastName) => $"{firstName} {lastName}",
    };
}
```

Сравнение объема кода

```
public readonly record struct PersonName(
  string FirstName, string LastName);
```

Сравнение объема кода

```csharp
[IsReadOnly]
public struct PersonName : IEquatable<PersonName>
{
    public string FirstName { get; init; }
    public string LastName { get; init; }

    public PersonName(string FirstName, string LastName)
    {
        this.FirstName = FirstName;
        this.LastName = LastName;
    }

    public override string ToString()
    {
        StringBuilder stringBuilder = new StringBuilder();
        stringBuilder.Append("PersonName");
        stringBuilder.Append(" { ");
        if (PrintMembers(stringBuilder))
        {
            stringBuilder.Append(' ');
        }
        stringBuilder.Append('}');
        return stringBuilder.ToString();
    }

    private bool PrintMembers(StringBuilder builder)
    {
        builder.Append("FirstName = ");
        builder.Append((object)FirstName);
        builder.Append(", LastName = ");
        builder.Append((object)LastName);
        return true;
    }

    public static bool operator !=(PersonName left, PersonName
      right)
    {
        return !(left == right);
    }

    public static bool operator ==(PersonName left, PersonName
      right)
    {
        return left.Equals(right);
    }

    public override int GetHashCode()
    {
        return
      EqualityComparer<string>.Default.GetHashCode(FirstName) *
      -1521134295 +
      EqualityComparer<string>.Default.GetHashCode(LastName);
    }

    public override bool Equals(object? Obj)
    {
        return obj is PersonName && Equals((PersonName)obj);
    }

    public bool Equals(PersonName other)
    {
        return
      EqualityComparer<string>.Default.Equals(FirstName,
      other.FirstName) &&
      EqualityComparer<string>.Default.Equals(LastName,
      other.LastName);
    }

    public void Deconstruct(out string FirstName, out string
      LastName)
    {
        FirstName = this.FirstName;
        LastName = this.LastName;
    }
}
```

Выводы

- ✓ Удобство
- ✓ Минимализм
- ✓ Синтаксис инициализации
- ✓ Init-only setters
- ✓ Структурное равенство
- ✓ Возможность Immutability
- ✓ System.Text.Json
- ✓ Newtonsoft.Json
- ✓ AutoMapper

- ✗ Mutability по умолчанию
- ✗ Shallow immutability
- ✗ Default value expression init
- ✗ Нет перегрузки Equals() и GetHashCode() для structs

Выводы

- Record structs – вместо structs
- Readonly record structs – вместо readonly structs
- Structs – подводные камни
- Immutable collections – пользуйтесь
- Class – Value Objects/Entities/ООП

Литература по Record structs

- Records (C# reference) https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/record
- Record structs specification https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-10.0/record-structs
- Parameterless constructors and field initializers https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct#parameterless-constructors-and-field-initializers

Литература по Record structs

- C# 10 - record struct Deep Dive & Performance Implications https://nietras.com/2021/06/14/csharp-10-record-struct/
- C# 10 Record Structs https://medium.com/general-thoughts/c-10-record-structs-bf73353ed7bc
- Using C# 10 outside .NET 6 https://github.com/dotnet/roslyn/discussions/47701#discussioncomment-1356495

Литература по Records

- 6 less popular facts about C# 9 records https://tooslowexception.com/6-less-popular-facts-about-c-9-records/
- Avoid C# 9 Record Gotchas https://khalidabuhakmeh.com/avoid-csharp-9-record-gotchas
- Илья Шипунов «C# 9 Records» https://github.com/DotNetRu/BrandBook/wiki/CSharp9-Records

Литература по Structs

- The ‘in’-modifier and the readonly structs in C#
  https://devblogs.microsoft.com/premier-developer/the-in-modifier-and-the-readonly-structs-in-c/
- Performance implications of default struct equality in C#
  https://devblogs.microsoft.com/premier-developer/performance-implications-of-default-struct-equality-in-c/
- Magic behind ValueType.Equals
  https://web.archive.org/web/20100714231838/http://blogs.msdn.com/b/xiangfan/archive/2008/09/01/magic-behind-valuetype-equals.aspx

Framework Design Guidelines

- Struct Design
  https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/struct
- Choosing Between Class and Struct
  https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/choosing-between-class-and-struct
- When to use record vs class vs struct
  https://stackoverflow.com/a/64828780

Спасибо за внимание

Вопросы?