.NET黄浦论坛
— . NET HUANGPU FORUM —

2024.NET技术沙龙

使用 C# 新特性构建整洁代码

演讲嘉宾：李卫涵

.NET黄浦论坛
— . NET HUANGPU FORUM —

目录

CONTENTS

可读

简洁

可维护

可扩展

高效

好协作

优质

易重构

Caller Info

nameof expression

Global Usings

Pattern Matching

Collection Expression

Params Collections

Filed Namespace/Types

More …

How Clean Code - Caller Info


```
    public static void CallerMemberNameTest()
    {
        // ...
        LogSuccess();
    }

    public static void CallerMemberNameTest2()
    {
        // ...
        LogSuccess();
    }

    private static void LogSuccess([CallerMemberName]string memberName = "")
    {
        // ...
        Console.WriteLine($"{memberName} invoked");
    }
```

```
    // ...
    Console.WriteLine($"{memberName} invoked");
    _successCounter ??= new Meter("CleanCSharpSamples")
        .CreateCounter<int>("invoke_success_counter", "{times}", "Method invoke success count");
    _successCounter.Add(1, new KeyValuePair<string, object?>("method", memberName));
}
```

How Clean Code - Caller Info


```
public static void CallerLocationTest()
{
    try
    {
        var a = 0;
        // ReSharper disable once IntDivisionByZero
        Console.WriteLine(1/a);
        LogSuccess();
    }
    catch (Exception e)
    {
        LogException(e);
    }
}

private static void LogException(Exception exception,
    [CallerMemberName]string memberName = "",
    [CallerFilePath]string filePath = "",
    [CallerLineNumber]int lineNumber = 0
    )
{
    Console.WriteLine($"{memberName} invoke exception, exception source:{filePath},
{lineNumber}:\n{exception.Message}");
}
```

How Clean Code - Caller Info


```
public static void CallerArgumentExpressionTest()
{
    try
    {
        object? name = "test";
        ThrowIfNull(name);

        name = null;
        ThrowIfNull(name);
    }
    catch (ArgumentNullException e)
    {
        WriteLine(e.Message);
    }

    LogInputExpression(1 + 1);

    var num1 = 1;
    var num2 = 2;
    LogInputExpression(num1 + num2);
}

private static object ThrowIfNull(object? obj, [CallerArgumentExpression(nameof(obj))] string name = "")
{
    if (obj is null)
        throw new ArgumentNullException(name);

    return obj;
}

private static void LogInputExpression(int input, [CallerArgumentExpression(nameof(input))] string name = "")
{
    Console.WriteLine($"input: {name}, {input}");
}
```

How Clean Code - Caller Info


```
public static void CallerArgumentExpressionTest()
{
    try
    {
        object? name = "test";
        ThrowIfNull(name);

        name = null;
        ThrowIfNull(name);
    }
    catch (ArgumentNullException e)
    {
        WriteLine(e.Message);
    }

    LogInputExpression(1 + 1);

    var num1 = 1;
    var num2 = 2;
    LogInputExpression(num1 + num2);
}

private static object ThrowIfNull(object? obj, [CallerArgumentExpression(nameof(obj))] string name = "")
{
    if (obj is null)
        throw new ArgumentNullException(name);

    return obj;
}

private static void LogInputExpression(int input, [CallerArgumentExpression(nameof(input))] string name = "")
{
    Console.WriteLine($"input expression: {name}, {nameof(input)}: {input}");
}
```

```
Value cannot be null. (Parameter 'name')
input: 1 + 1, 2
input: num1 + num2, 3
```

How Clean Code - Caller Info


```
/// <summary>Throws an <see cref="ArgumentNullException"/> if <paramref name="argument"/> is null.</summary>
/// <param name="argument">The reference type argument to validate as non-null.</param>
/// <param name="paramName">The name of the parameter with which <paramref name="argument"/> corresponds.</param>
public static void ThrowIfNull([NotNull] object? argument, [CallerArgumentExpression(nameof(argument))] string? paramName = null)
{
    if (argument is null)
    {
        Throw(paramName);
    }
}
```

```
[DoesNotReturn]
internal static void Throw(string? paramName) =>
    throw new ArgumentNullException(paramName);
```

How Clean Code - nameof


```
public static void Run(string[] args)
{
    Console.WriteLine("Executing in NameOfSamples...");
    Console.WriteLine($"Executing in {nameof(Run)}...");

    var title = "Engineer";
    Console.WriteLine($"{nameof(title)}: {title}");

    Console.WriteLine($"Arguments: {nameof(args)}-{string.Join(", ", args)}");
}
```

```
dotnet-exec .\NameOfSamples.cs
Executing in NameOfSamples...
Executing in Run...
title: Engineer
Arguments: args-
```

```
[return: NotNullIfNotNull(nameof(name))]
private static string? MoreNameOfUsage([NotNullIfNotNull(nameof(description))]string? name, string? description)
{
    if (!string.IsNullOrEmpty(description) && name is null)
    {
        throw new ArgumentException("The name cannot be null when description is not null or empty.");
    }
    return name;
}
```

```
public static class NameOfSamples
{
    private const string ArgmentName = nameof(ArgmentName);
    private const string ArgumentName = nameof(ArgumentName);
    private static readonly string AnotherArgumentName = nameof(AnotherArgumentName);
}
```

小心重构使用 nameof 的常量

How Clean Code - Global Using


```
// global using samples
global using System;
global using static System.Console;
global using Runtime = System.Environment;
global using MyCache = System.Collections.Co
    System.Collections.Concurrent.Concurrent
```

```
<ItemGroup>
  <Using Include="$(RootNamespace)"/>
  <Using Include="WeihanLi.Common.Helpers"/>
  <Using Include="WeihanLi.Common.Helpers.InvokeHelper" Static="true"/>
  <Using Include="WeihanLi.Common.Logging"/>
  <Using Include="System.IO.File" Alias="MyFile"/>
  <Using Remove="System.Net.Http" />
</ItemGroup>
```

```
<PropertyGroup>
    <LangVersion>latest</LangVersion>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <PackageIcon>icon.png</PackageIcon>
```

How Clean Code - Global Using


```
using System.Collections.Concurrent;

// file-scoped namespace
namespace CleanCSharpSamples;

public static class GlobalUsingsSamples
{
    private static readonly ConcurrentDictionary<string,
ConcurrentDictionary<string, string>> Cache
        = new ConcurrentDictionary<string, ConcurrentDictionary<string,
string>>();
    private static readonly MyCache Cache1 = new();

    public static void UsingSamples()
    {
        {
            System.Console.WriteLine("Hello Static Using!");

 System.Console.WriteLine(System.Environment.Version.ToString());
        }

        {
            // global using static
            WriteLine("Hello Static Using!");

            // global using alias
            WriteLine(Runtime.Version.ToString());
        }
    }
}
```

```
object obj = new Cat("Mi");

var cat0 = obj as Cat;
if (cat0 ≠ null)
{
    Console.WriteLine(cat0.Name);
}

if (obj is Cat cat)
{
    Console.WriteLine(cat.Name);
}
```

```
object obj = new Cat("Mi");

switch (obj)
{
    case Cat c:
        Console.WriteLine($"Hello Cat: {c.Name}");
        break;

    case Dog d:
        Console.WriteLine($"Hello Dog: {d.Name}");
        break;

    case Mouse m:
        Console.WriteLine($"Hello Mouse: {m.Name}");
        break;
}
```

```
file abstract record Animal(string Name);
file sealed record Cat(string Name) : Animal(Name);
file sealed record Dog(string Name) : Animal(Name);
file sealed record Mouse(string Name) : Animal(Name);
```

How Clean Code - Pattern Matching


```
var a :{FirstName,Age,...} = new
{
    FirstName = "Mike",
    Age = 10,
    Job = new
    {
        Current = new
        {
            Title = "Engineer",
            Manager = new
            {
                FirstName = "John"
            }
        },
        Histories = new[]
        {
            new
            {
                Title = "Engineer I"
            }
        }
    }
};
```

```
if (a ≠ null
    && a.Age > 3
    && a.Job ≠ null
    && a.Job.Current ≠ null
    && a.Job.Histories ≠ null
    && a.Job.Histories.Length > 0
    && a.Job.Current.Title == "Engineer"
    && a.Job.Current.Manager ≠ null
    && a.Job.Current.Manager.FirstName == "John"
    )
{
    Console.WriteLine($"{a.FirstName} is {a.Age} years old {a.Job.Current.Title}");
}
```

```
if (a is
    {
        Age : > 3,
        Job:
        {
            Histories.Length: > 0,
            Current:
            {
                Title: "Engineer",
                Manager.FirstName: "John"
            }
        }
    })
{
    Console.WriteLine($"{a.FirstName} is {a.Age} years old {a.Job.Current.Title}");
}
```

How Clean Code - Pattern Matching


```
var code = "Success";
var returnCode = Enum.Parse<ReturnCode>(code, true);
HttpStatusCode statusCode;
switch (returnCode)
{
    case ReturnCode.Success:
        statusCode = HttpStatusCode.OK;
        break;

    case ReturnCode.BadRequest:
        statusCode = HttpStatusCode.BadRequest;
        break;

    case ReturnCode.Unauthorized:
        statusCode = HttpStatusCode.Unauthorized;
        break;

    case ReturnCode.Forbidden:
        statusCode = HttpStatusCode.Forbidden;
        break;

    case ReturnCode.InternalError:
        statusCode = HttpStatusCode.InternalServerError;
        break;

    default:
        throw new ArgumentException("Invalid return code");
}
Console.WriteLine(statusCode);
```

How Clean Code - Pattern Matching


```
var code = "Success";
var returnCode = Enum.Parse<ReturnCode>(code, true);
var statusCode = returnCode switch
{
    ReturnCode.Success ⇒ HttpStatusCode.OK,
    ReturnCode.BadRequest ⇒ HttpStatusCode.BadRequest,
    ReturnCode.Unauthorized ⇒ HttpStatusCode.Unauthorized,
    ReturnCode.Forbidden ⇒ HttpStatusCode.Forbidden,
    ReturnCode.InternalError ⇒ HttpStatusCode.InternalServerError,
    _ ⇒ throw new ArgumentException("Invalid return code")
};
Console.WriteLine(statusCode);
```

```
var text = "Hello World";
var name = text switch
{
    "Hello World" ⇒ "World",
    "Hello CSharp" or "Hello C#" ⇒ "C#",
    "Hello dotnet" ⇒ "dotnet",
    _ ⇒ "Unknown"
};
Console.WriteLine(name);
```

```
var score = int.Parse("66");
var grade = score switch
{
    < 30 ⇒ "D-",
    < 60 ⇒ "D",
    < 80 ⇒ "C",
    < 90 ⇒ "B",
    < 95 ⇒ "A",
    _ ⇒ "A+",
};
Console.WriteLine(grade);
```

```
object obj = new Cat("Mi");

if (obj == null)
{
}
if (null == obj)
{
}
if (obj is null)
{
}

if (obj ≠ null)
{
}
if (obj is not null)
{
}
```

```
private static Point Transform(Point point)
    => point switch
{
    { X: 0, Y: 0 }                          => new Point(0, 0),
    { X: var x, Y: var y } when x < y => new Point(x + y, y),
    { X: var x, Y: var y } when x > y => new Point(x - y, y),
    { X: var x, Y: var y }                  => new Point(2 * x, 2 * y),
};


// primary constructor
public readonly ref struct Point(int x, int y)
{
    public int X { get; } = x;
    public int Y { get; } = y;
}
```

How Clean Code - Pattern Matching


```
var array = new[]
{
    1,2,3,4,5
};

if (array is [1, ..])
{
    Console.WriteLine("The first one is 1");
}
if (array is [.., 5])
{
    Console.WriteLine("The last one is 5");
}
if (array is [_, _, 3, ..])
{
    Console.WriteLine("The third one is 3");
}
if (array is [.., 4, _])
{
    Console.WriteLine("The second to last one is 4");
}
if (array is [1, 2, 3, ..])
{
    Console.WriteLine("The sequence starts with 1,2,3");
}
if (array is [.., 3, 4, 5])
{
    Console.WriteLine("The sequence ends with 3,4,5");
}
if (array is [.., 3, 4, _])
{
    Console.WriteLine("The sequence ends with 3,4,_");
}
```

How Clean Code - Pattern Matching


```
var objects = new object[]
{
    1, "2", 3.0, "4", Guid.NewGuid()
};

if (objects is [1, "2", ..])
{
    Console.WriteLine($"{objects[0]},{objects[1]}");
}
if (objects is [.., string str, Guid guid])
{
    Console.WriteLine($"{str},{guid:N}");
}

var result = objects switch
{
    [1, ..] => 1,
    [_, "2", ..] => 2,
    [_, _, double val, ..] => (int)val,
    [.., ""] => 4,
    [.., string strVal, Guid _] => Convert.ToInt32(strVal),
    _ => -1
};
Console.WriteLine(result);
```

```
int[] numArray = [1, 2, 3];
HashSet<int> numSet = [1, 2, 3];
List<int> numList = [1, 2, 3];
Span<char> charSpan = ['H', 'e', 'l', 'l', 'o'];
ReadOnlySpan<string> stringSpan = ["Hello", "World"];
ImmutableArray<string> immutableArray = ["Hello", "World"];


int[] nums = [1, 1, ..numArray, 2, 2];
Console.WriteLine(string.Join(",", nums));

int[] row0 = [1, 2, 3];
int[] row1 = [4, 5, 6];
int[] row2 = [7, 8, 9];
int[] single = [..row0, ..row1, ..row2];
foreach (var element in single)
{
    Console.Write($"{element}, ");
}
```

```
dotnet-exec .\CollectionExpressionSample.cs
1,1,1,2,3,2,2
1, 2, 3, 4, 5, 6, 7, 8, 9,
```

```
IEnumerable<string> enumerable = ["Hello", "dotnet"];
Console.WriteLine(enumerable.GetType());
IReadOnlyCollection<string> readOnlyCollection = ["Hello", "dotnet"];
Console.WriteLine(readOnlyCollection.GetType());
ICollection<string> collection = ["Hello", "dotnet"];
Console.WriteLine(collection.GetType());

IReadOnlyList<string> readOnlyList = ["Hello", "dotnet"];
Console.WriteLine(readOnlyList.GetType());
IList<string> list = ["Hello", "dotnet"];
Console.WriteLine(list.GetType());

IEnumerable<string> emptyEnumerable = [];
Console.WriteLine(emptyEnumerable.GetType());
IReadOnlyCollection<string> emptyReadonlyCollection = [];
Console.WriteLine(emptyReadonlyCollection.GetType());
ICollection<string> emptyCollection = [];
Console.WriteLine(emptyCollection.GetType());
IReadOnlyList<string> emptyReadOnlyList = [];
Console.WriteLine(emptyReadOnlyList.GetType());
IList<string> emptyList = [];
Console.WriteLine(emptyList.GetType());
```

How Clean Code - Params Collections


```
private static void ParamsArrayMethod(params int[] array)
{
    foreach (var item in array)
    {
        Console.WriteLine(item);
    }
}

private static void ParamsReadOnlySpanMethod(params ReadOnlySpan<int> collection)
{
    foreach (var item in collection)
    {
        Console.WriteLine(item);
    }
}

private static void ParamsSpanMethod(params Span<int> collection)
{
    foreach (var item in collection)
    {
        Console.WriteLine(item);
    }
}

private static void ParamsListMethod(params List<int> list)
{
    foreach (var item in list)
    {
        Console.WriteLine(item);
    }
}

private static void ParamsEnumerableMethod(params IEnumerable<int> array)
{
    foreach (var item in array)
    {
        Console.WriteLine(item);
    }
}
```

```
public static void Run()
{
    ParamsArrayMethod(1, 2, 3);
    ParamsListMethod(1, 2, 3);
    ParamsEnumerableMethod(1, 2, 3);
    ParamsSpanMethod(1, 2, 3);
    ParamsReadOnlySpanMethod(1, 2, 3);
}
```

How Clean Code - Filed Namespace


```
namespace CleanCSharpSamples
{
    public class FileScopedNamespaceSample
    {
        public static void MainTest()
        {
            if (OperatingSystem.IsWindows())
            {
                Console.WriteLine("Running on Windows");
            }
            else
            {
                Console.WriteLine("Running on Non-Windows");
            }
        }
    }
}
```

```
namespace CleanCSharpSamples;

public class FileScopedNamespaceSample
{
    public static void MainTest()
    {
        if (OperatingSystem.IsWindows())
        {
            Console.WriteLine("Running on Windows");
        }
        else
        {
            Console.WriteLine("Running on Non-Windows");
        }
    }
}
```

file-scoped namespace 文件范围命名空间

How Clean Code - File Local Types


```
internal sealed class FileLocalTypes
{
  private interface IEnvironment
  {
      string? GetEnvVal(string name);
  }
  private sealed class RuntimeEnvironment : IEnvironment
  {
      public string? GetEnvVal(string name) => Environment.GetEnvironmentVariable(name);
  }

  private sealed class Holder
  {
    private sealed class MockEnvironment : IEnvironment
    {
      public string? GetEnvVal(string name) => name;
    }
  }
}
```

How Clean Code - File Local Types


```
// file types
file interface IEnvironment
{
    string? GetEnvVal(string name);
}

file sealed class RuntimeEnvironment : IEnvironment
{
    public string? GetEnvVal(string name) => Environment.GetEnvironmentVariable(name);
}

file sealed class MockEnvironment : IEnvironment
{
    public string? GetEnvVal(string name) => name;
}
```

How Clean Code - Raw String Literals

```
var html= @"<div style=""font-size:1.2em""><p>Header Text</p></div>";
Console.WriteLine(html);

html = """<div style="font-size:1.2em"><p>Header Text</p></div>""";
Console.WriteLine(html);

var text = "Header Text";
html = $"""<div style="font-size:1.2em"><p>{text}</p></div>""";
Console.WriteLine(html);

var anotherText = """"Header """ Text """";
Console.WriteLine(anotherText);

var sql = """
          SELECT [Id], [Name], [Description]
          FROM dbo.tbl_Users WITH(NOLOCK)
          WHERE [IsDeleted] = 0
          """;
Console.WriteLine(sql);

sql = """
      SELECT [Id], [Name], [Description]
      FROM dbo.tbl_Users WITH(NOLOCK)
      WHERE [IsDeleted] = 0
""";
Console.WriteLine(sql);
```

How Clean Code - Raw String Literals


```
var json = """{"name":"Mike","age":10}""";
Console.WriteLine(json);
json = """
       {
         "name": "Mike",
         "age": 10
       }
       """;
Console.WriteLine(json);

var name = "Mike";
json = $"""
       {
         "name": "{{name}}",
         "age": 10
       }
       """;
Console.WriteLine(json);
json = $$$"""
         {
           "name": "{{{name}}}",
           "title": "{{I}}",
           "age": 10
         }
         """;
Console.WriteLine(json);
```

How Clean Code - Record Types


```
public static class RecordSample
{
    public static void Run()
    {
        var student = new Student { Id = 1, Name = "Mike" };
        var student2 = new Student { Id = 1, Name = "Mike" };
        // output true
        Console.WriteLine(student == student2);

        Console.WriteLine(student);
        var updatedStudent = student with { Name = "Ming" };
        Console.WriteLine(updatedStudent);
        Console.WriteLine(new Point(1, 2));
    }
}

file record Student
{
    // required members
    // init only setter
    public required int Id { get; init; }
    public required string Name { get; init; }
}

file record struct Point(int X, int Y);
```

```
└☁ dotnet-exec .\RecordSample.cs
True
Student { Id = 1, Name = Mike }
Student { Id = 1, Name = Ming }
Point { X = 1, Y = 2 }
```

```
file sealed class AppService1
{
    private readonly IEnvironment _environment;

    public AppService1(IEnvironment environment)
    {
        _environment = environment;
    }
    public void Test()
    {
        Console.WriteLine(_environment.GetEnvVal(nameof(Test)));
    }
}

// primary constructor
file sealed class AppService(IEnvironment environment)
{
    public void Test()
    {
        Console.WriteLine(environment.GetEnvVal(nameof(Test)));
    }
}
```

How Clean Code - UTF-8 String Literal

```
var lineEnd = "\r\n"u8;
using var responseStream = new MemoryStream();
responseStream.Write("id,name"u8);
responseStream.Write(lineEnd);
var list = new List<Student>()
{
    new()
    {
        Id = 1,
        Name = "Ming"
    },
    new()
    {
        Id = 2,
        Name = "Mike"
    }
};
foreach (var student in list)
{
    var lineBytes = Encoding.UTF8.GetBytes($"{student.Id},
{student.Name}");
    responseStream.Write(lineBytes);
    responseStream.Write(lineEnd);
}
```

Index & Range

```
Student[] array =
[
    new()
    {
        Id = 1,
        Name = "Ming"
    },
    new()
    {
        Id = 2,
        Name = "Mike"
    },
    new()
    {
        Id = 3,
        Name = "Jane"
    },
];
Console.WriteLine(array[^1].Name);
foreach (var student in array[1..])
{
    Console.WriteLine($"{student.Id} {student.Name}");
}
```

```
var hello = "Hello World";
var separator = hello.IndexOf(' ');
if (separator > 0)
{
    var first = hello[..separator];
    var second = hello[(separator + 1)..];
    Console.WriteLine($"{first} {second}");
}
```

Editorconfig

- 代码风格约定配置

Rider/Resharper

- 智能提示

SonarScanner/Qodana

- 静态代码分析

Tools - editorconfig


```
###############################
# .NET Coding Conventions     #
###############################
[*.{cs,vb}]
# Organize usings
dotnet_sort_system_directives_first = true
# this. preferences
dotnet_style_qualification_for_field = true:error
dotnet_style_qualification_for_property = true:error
dotnet_style_qualification_for_method = true:error
dotnet_style_qualification_for_event = true:error
# Language keywords vs BCL types preferences
dotnet_style_predefined_type_for_locals_parameters_members = true:su
dotnet_style_predefined_type_for_member_access = true:suggestion
# Parentheses preferences
dotnet_style_parentheses_in_arithmetic_binary_operators = always_for
dotnet_style_parentheses_in_relational_binary_operators = always_for
dotnet_style_parentheses_in_other_binary_operators = always_for_clar
dotnet_style_parentheses_in_other_operators = never_if_unnecessary:s
# Modifier preferences
dotnet_style_require_accessibility_modifiers = for_non_interface_mem
dotnet_style_readonly_field = true:warning
# Expression-level preferences
dotnet_style_object_initializer = true:suggestion
dotnet_style_collection_initializer = true:suggestion
dotnet_style_explicit_tuple_names = true:suggestion
dotnet_style_null_propagation = true:suggestion
dotnet_style_coalesce_expression = true:suggestion
dotnet_style_prefer_is_null_check_over_reference_equality_method = t
```

```
[*.cs]
# Note: these settings cause "dotnet format" to fix the code. You should review each change if you uses "dotnet format".
dotnet_diagnostic.RCS1036.severity = warning # Remove unnecessary blank line.
dotnet_diagnostic.RCS1037.severity = warning # Remove trailing white-space.
dotnet_diagnostic.RCS1097.severity = warning # Remove redundant 'ToString' call.
dotnet_diagnostic.RCS1138.severity = warning # Add summary to documentation comment.
dotnet_diagnostic.RCS1139.severity = warning # Add summary element to documentation comment.
dotnet_diagnostic.RCS1168.severity = warning # Parameter name 'foo' differs from base name 'bar'.
dotnet_diagnostic.RCS1175.severity = warning # Unused 'this' parameter 'operation'.
dotnet_diagnostic.RCS1192.severity = warning # Unnecessary usage of verbatim string literal.
dotnet_diagnostic.RCS1194.severity = warning # Implement exception constructors.
dotnet_diagnostic.RCS1211.severity = warning # Remove unnecessary else clause.
dotnet_diagnostic.RCS1214.severity = warning # Unnecessary interpolated string.
dotnet_diagnostic.RCS1225.severity = warning # Make class sealed.
dotnet_diagnostic.RCS1232.severity = warning # Order elements in documentation comment.

# Commented out because `dotnet format` change can be disruptive.
# dotnet_diagnostic.RCS1085.severity = warning # Use auto-implemented property.

# Commented out because `dotnet format` removes the xmldoc element, while we should add the missing documentation instead.
# dotnet_diagnostic.RCS1228.severity = warning # Unused element in documentation comment.

# Diagnostics elevated as warnings
dotnet_diagnostic.CA1000.severity = warning # Do not declare static members on generic types
dotnet_diagnostic.CA1031.severity = warning # Do not catch general exception types
dotnet_diagnostic.CA1050.severity = warning # Declare types in namespaces
dotnet_diagnostic.CA1063.severity = warning # Implement IDisposable correctly
dotnet_diagnostic.CA1064.severity = warning # Exceptions should be public
dotnet_diagnostic.CA1416.severity = warning # Validate platform compatibility
dotnet_diagnostic.CA1508.severity = warning # Avoid dead conditional code
dotnet_diagnostic.CA1852.severity = warning # Sealed classes
```

Tools - Rider


```
file sealed class AppService1
{
    private readonly IEnvironment _environment;

    public AppService1(IEnvironment environment)
    {
        _environment =
    }
    1 usage
    public void Test()
    {
        Console.WriteLine(_environment.GetEnvVal(nameof(Test)));
    }
```

Convert into primary constructor

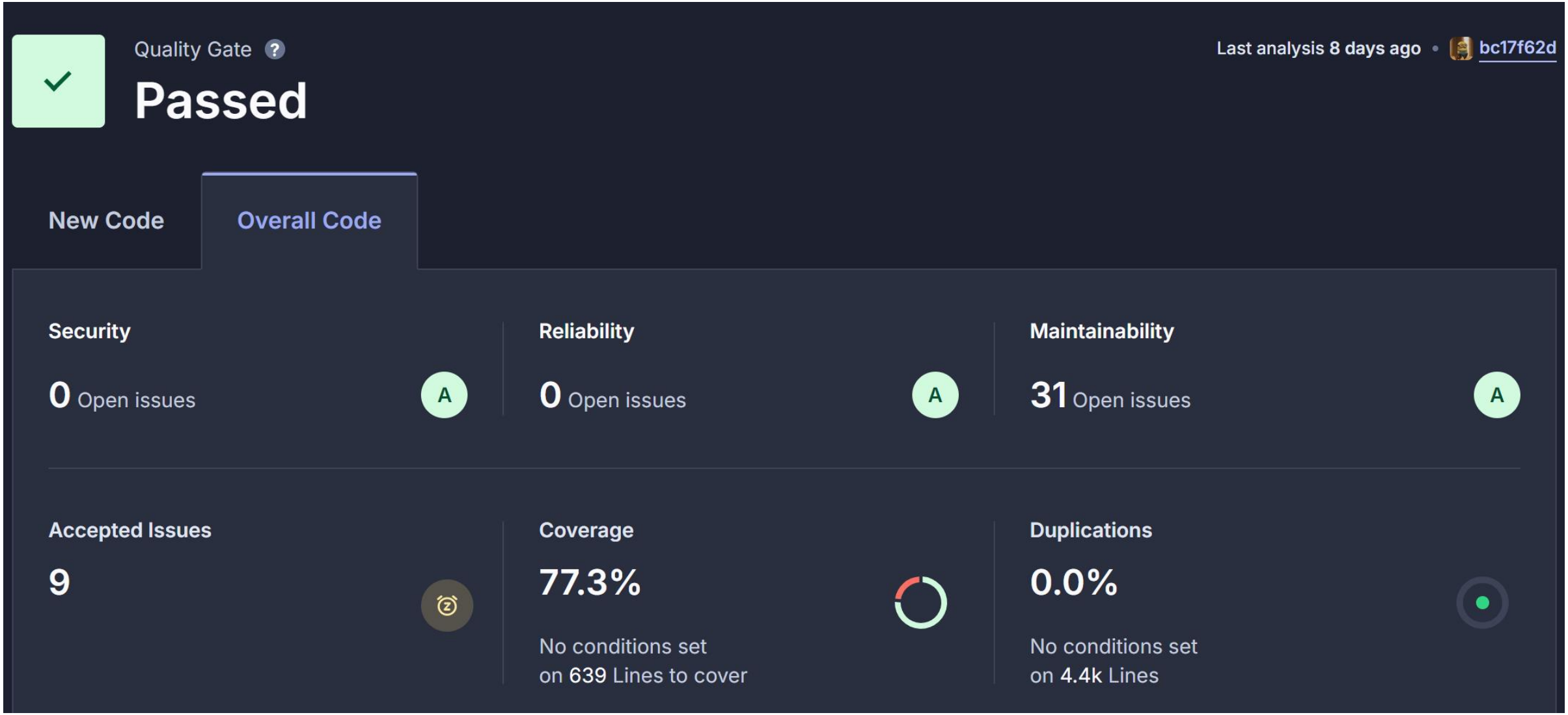
```
public AppService1(IEnvironment environment)
 in class CleanCSharpSamples.AppService1
```

Tools - Rider


```
try
{
    // ...
    WriteLine("success");
}
catch (Exception e)
{
    var timeoutException = e as TimeoutException;
    var operationCanceledExcep                    CanceledException;
    if (timeoutException != nu                    edException != null)
    {
        throw;
    }
```

Use pattern matching

Tools - SonarCloud



Adaptability

Refactor this method to reduce its Cognitive Complexity from 21 to the 15 allowed.

brain-overload +

Open · Not assigned · Maintainability · Code Smell · Critical

6min effort • 3 months ago

Intentionality

Loop should be simplified by calling Select(item => item.Identity))

No tags +

Open · Not assigned · Maintainability · Code Smell · Minor

5min effort • 3 months ago

Adaptability

Refactor this method to reduce its Cognitive Complexity from 32 to the 15 allowed.

brain-overload +

Open · Not assigned · Maintainability · Code Smell · Critical

6min effort • 9 months ago

Intentionality

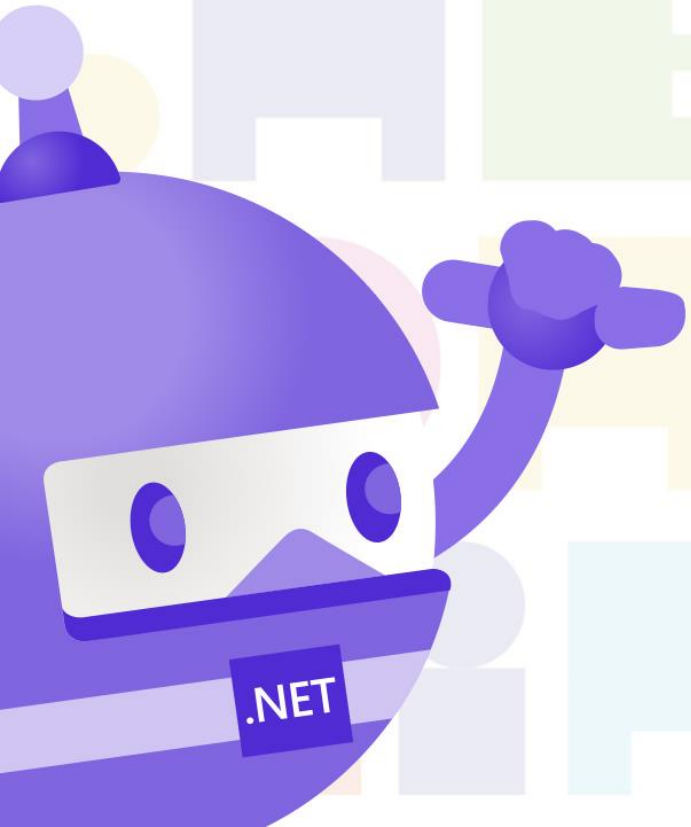
Extract this nested ternary operation into an independent statement.

confusing +

Open · Not assigned · Maintainability · Code Smell · Major

References

- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/nameof
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/is
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/operators/switch-expression
- https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/functional/pattern-matching
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/attributes/caller-information
- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/
- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/file
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-11.0/file-local-types
- https://devblogs.microsoft.com/dotnet/csharp-primary-constructors-refactoring/
- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/primary-constructors
- https://devblogs.microsoft.com/dotnet/refactor-your-code-with-collection-expressions/
- https://devblogs.microsoft.com/dotnet/announcing-csharp-12/
- https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/patterns-objects
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/proposals/csharp-10.0/file-scoped-namespaces
- https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/using-directive
- https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/code-style-rule-options
- https://learn.microsoft.com/en-us/dotnet/csharp/tutorials/records
- https://learn.microsoft.com/en-us/dotnet/csharp/tutorials/ranges-indexes
- https://github.com/WeihanLi/SamplesInPractice/tree/main/CleanCSharpSamples



Keep Coding

Clean Code

感谢观看

多元化社区 | 大咖面对面 | 技术交流圈 | 编程挑战赛 | 求职招聘圈　尽在黄浦论坛

.NET黄浦论坛
—. NET HUANGPU FORUM—

2024.NET技术沙龙