Лекция. Работа со строками

МДК 01.04. Системное программирование

План лекции:

1. **Введение в строки в .NET**
    - Что такое строка в .NET
    - Особенности строк как ссылочных типов
    - Иммутability (неизменяемость) строк
2. **Создание и инициализация строк**
    - Различные способы создания строк
    - Строковые литералы vs. объекты String
    - Использование конструкторов
3. **Основные операции со строками**
    - Конкатенация и интерполяция
    - Сравнение строк
    - Поиск и извлечение подстрок
4. **Форматирование строк**
    - Метод String.Format()
    - Интерполяция строк
    - Форматирование чисел и дат
5. **Методы работы со строками**
    - Модификация строк
    - Разделение и объединение
    - Работа с пустыми строками
6. **StringBuilder - эффективная работа со строками**
    - Зачем нужен StringBuilder
    - Основные методы и свойства
    - Сравнение производительности
7. **Регулярные выражения**
    - Базовые понятия и синтаксис
    - Класс Regex и его методы
    - Практические примеры

---

1. Введение в строки в .NET

Что такое строка в .NET? В .NET строка представляет собой последовательность символов Unicode. Тип `string` (который является псевдонимом для `System.String`) - это ссылочный тип, но с семантикой, похожей на типы значений.

```
// string - псевдоним для System.String
string name = "Иван";
System.String fullName = "Иван Петров"; // Эквивалентно
```

Иммутабельность (неизменяемость) строк Одна из ключевых особенностей строк в .NET - их неизменяемость. Это означает, что после создания строку нельзя изменить. Любая операция, которая якобы изменяет строку, на самом деле создает новую строку.

```
string original = "Hello";
string modified = original.ToUpper(); // Создается НОВАЯ строка

Console.WriteLine(original);  // "Hello"
Console.WriteLine(modified);  // "HELLO"

// Демонстрация иммутабельности
string s1 = "test";
string s2 = s1;
s1 += " modified";
Console.WriteLine(s1); // "test modified"
Console.WriteLine(s2); // "test" - s2 не изменилась!
```

Преимущества иммутабельности:

- Потокобезопасность
- Предсказуемое поведение
- Возможность кэширования хеш-кодов

---

2. Создание и инициализация строк

Различные способы создания строк:

```
// 1. Строковый литерал
string str1 = "Hello World";

// 2. Использование конструктора из массива char
char[] charArray = {'H', 'e', 'l', 'l', 'o'};
string str2 = new string(charArray);

// 3. Конструктор с повторением символа
string str3 = new string('*', 10); // "**********"

// 4. Из части массива символов
```

```
char[] chars = {'a', 'b', 'c', 'd', 'e'};
string str4 = new string(chars, 1, 3); // "bcd"

// 5. Пустая строка
string empty1 = "";
string empty2 = string.Empty; // Предпочтительный способ
```

Строковые литералы с специальными символами:

```
// Экранированные последовательности
string path = "C:\\Windows\\System32"; // Обратный слеш
string quote = "Он сказал: \"Привет!\""; // Кавычки
string newLine = "Первая строка\nВторая строка"; // Перевод строки
string tab = "Имя:\tЗначение"; // Табуляция

// Дословные строки (verbatim strings)
string verbatimPath = @"C:\Windows\System32"; // Не нужно экранировать \
string multiLine = @"Это
многострочная
строка";

// Интерполированные строки
string name = "Мария";
int age = 25;
string interpolated = $"{name} имеет возраст {age} лет";
```

---

3. Основные операции со строками

Конкатенация строк:

```
// Оператор +
string firstName = "Иван";
string lastName = "Петров";
string fullName = firstName + " " + lastName;

// Метод Concat
string result = string.Concat(firstName, " ", lastName);

// Метод Join для объединения массива
string[] words = {"Я", "изучаю", "C#"};
string sentence = string.Join(" ", words); // "Я изучаю C#"

// Множественная конкатенация (неэффективно!)
string numbers = "";
for (int i = 0; i < 5; i++)
{
    numbers += i.ToString(); // Создается новая строка на каждой итерации!
}
```

Сравнение строк:

```
string str1 = "Hello";
string str2 = "HELLO";
string str3 = "Hello";

// Оператор == (учитывает регистр по умолчанию)
bool equal1 = (str1 == str3); // true
bool equal2 = (str1 == str2); // false

// Метод Equals
bool equal3 = str1.Equals(str3); // true
bool equal4 = str1.Equals(str2, StringComparison.OrdinalIgnoreCase); // true

// Сравнение с учетом культуры
string strA = "straße";
string strB = "strasse";
bool equal5 = strA.Equals(strB, StringComparison.InvariantCulture); // true

// Метод Compare
int result1 = string.Compare(str1, str2); // отрицательное число (str1 < str2)
int result2 = string.Compare(str1, str2, StringComparison.OrdinalIgnoreCase); // 0
```

Поиск и извлечение подстрок:

```
string text = "Программирование на C# - это интересно!";

// Поиск индекса
int index1 = text.IndexOf("C#"); // 19
int index2 = text.IndexOf("Java"); // -1 (не найдено)
int index3 = text.LastIndexOf(" "); // 32

// Проверка наличия
bool contains = text.Contains("интересно"); // true
bool startsWith = text.StartsWith("Программирование"); // true
bool endsWith = text.EndsWith("!"); // true

// Извлечение подстрок
string substring1 = text.Substring(19, 2); // "C#"
string substring2 = text.Substring(22); // " - это интересно!"

// Извлечение по диапазону (C# 8.0+)
Range range = 0..14;
string substring3 = text[range]; // "Программирование"
```

---

4. Форматирование строк

Метод String.Format():

```
string name = "Анна";
int age = 30;
double salary = 45000.50;

// Простое форматирование
string result1 = string.Format("Имя: {0}, Возраст: {1}", name, age);

// Форматирование чисел
string result2 = string.Format("Зарплата: {0:C}", salary); // "Зарплата: 45 000,50
₽"
string result3 = string.Format("Число: {0:N2}", 1234.567); // "Число: 1 234,57"
string result4 = string.Format("Процент: {0:P}", 0.123); // "Процент: 12,30 %"

// Форматирование дат
DateTime now = DateTime.Now;
string date1 = string.Format("Дата: {0:d}", now); // "Дата: 01.01.2023"
string date2 = string.Format("Время: {0:t}", now); // "Время: 14:30"
string date3 = string.Format("Полная дата: {0:F}", now); // "Полная дата: 1 января
2023 г. 14:30:00"
```

Интерполяция строк (C# 6.0+):

```
string product = "Ноутбук";
decimal price = 75000.99m;
int quantity = 3;

// Базовая интерполяция
string message = $"{product} стоит {price:C}";

// Интерполяция с выражениями
string order = $"{quantity} × {product} = {price * quantity:C}";

// Интерполяция с форматированием
string formatted = $"Цена: {price:C2} | Дата: {DateTime.Now:yyyy-MM-dd}";

// Выравнивание в интерполированных строках
string aligned = $"{product,-15} {price,10:C} {quantity,5}";
```

---

5. Методы работы со строками

Модификация строк:

```
string text = "   Hello World!   ";

// Удаление пробелов
```

```
string trimmed = text.Trim();        // "Hello World!"
string trimStart = text.TrimStart(); // "Hello World!   "
string trimEnd = text.TrimEnd();     // "   Hello World!"

// Замена
string replaced = text.Replace("World", "C#"); // "   Hello C#!   "

// Изменение регистра
string upper = text.ToUpper(); // "   HELLO WORLD!   "
string lower = text.ToLower(); // "   hello world!   "

// Вставка и удаление
string original = "Hello!";
string inserted = original.Insert(5, " World"); // "Hello World!"
string removed = original.Remove(2, 3); // "He!"
```

Разделение и объединение:

```
// Разделение строки
string data = "Иван;Петров;30;Разработчик";
string[] parts = data.Split(';');
// parts = ["Иван", "Петров", "30", "Разработчик"]

// Разделение с несколькими разделителями
string sentence = "Я изучаю C#, Java и Python";
string[] words = sentence.Split(new char[] {' ', ',', '.'},
                                StringSplitOptions.RemoveEmptyEntries);

// Объединение массива в строку
string[] languages = {"C#", "Java", "Python"};
string joined = string.Join(", ", languages); // "C#, Java, Python"
```

Работа с пустыми строками:

```
string empty = "";
string whitespace = "   ";
string nullString = null;
string normal = "Text";

// Проверка на пустоту
bool isEmpty1 = string.IsNullOrEmpty(empty); // true
bool isEmpty2 = string.IsNullOrEmpty(nullString); // true
bool isEmpty3 = string.IsNullOrEmpty(whitespace); // false
bool isEmpty4 = string.IsNullOrEmpty(normal); // false

// Проверка на пустоту или пробелы
bool isBlank1 = string.IsNullOrWhiteSpace(empty); // true
bool isBlank2 = string.IsNullOrWhiteSpace(whitespace); // true
bool isBlank3 = string.IsNullOrWhiteSpace(normal); // false
```

---

6. StringBuilder - эффективная работа со строками

Зачем нужен StringBuilder: Когда выполняется множество операций модификации строк, использование обычных строк становится неэффективным из-за их неизменяемости. StringBuilder решает эту проблему.

```
using System.Text;

// Неэффективный способ
string result = "";
for (int i = 0; i < 1000; i++)
{
    result += i.ToString(); // Создается 1000 новых строк!
}

// Эффективный способ с StringBuilder
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 1000; i++)
{
    sb.Append(i.ToString()); // Изменяется существующий объект
}
string efficientResult = sb.ToString();
```

Основные методы и свойства StringBuilder:

```
StringBuilder sb = new StringBuilder();

// Добавление содержимого
sb.Append("Hello");
sb.AppendLine(" World!"); // Добавляет перевод строки
sb.AppendFormat("Сегодня: {0:dd.MM.yyyy}", DateTime.Now);

// Вставка и удаление
sb.Insert(5, " Beautiful"); // "Hello Beautiful World!"
sb.Remove(5, 10); // Удаляет " Beautiful"

// Замена
sb.Replace("World", "C#");

// Доступ к отдельным символам
sb[0] = 'h'; // Изменяет первый символ

// Получение результата
string result = sb.ToString();

// Очистка
sb.Clear();
```

```
// Работа с емкостью
StringBuilder sb2 = new StringBuilder(100); // Начальная емкость 100 символов
Console.WriteLine($"Емкость: {sb2.Capacity}, Длина: {sb2.Length}");
```

Сравнение производительности:

```
using System.Diagnostics;

// Тест с обычными строками
var stopwatch1 = Stopwatch.StartNew();
string s = "";
for (int i = 0; i < 10000; i++)
{
    s += i.ToString();
}
stopwatch1.Stop();

// Тест с StringBuilder
var stopwatch2 = Stopwatch.StartNew();
StringBuilder sb = new StringBuilder();
for (int i = 0; i < 10000; i++)
{
    sb.Append(i.ToString());
}
string result = sb.ToString();
stopwatch2.Stop();

Console.WriteLine($"String: {stopwatch1.ElapsedMilliseconds} ms");
Console.WriteLine($"StringBuilder: {stopwatch2.ElapsedMilliseconds} ms");
```

---

7. Регулярные выражения

Базовые понятия и синтаксис:

```
using System.Text.RegularExpressions;

// Проверка соответствия шаблону
string pattern = @"^\d{3}-\d{2}-\d{4}$"; // Формат SSN: 123-45-6789
string input = "123-45-6789";

bool isMatch = Regex.IsMatch(input, pattern); // true

// Поиск всех совпадений
string text = "Телефоны: +7-123-456-7890, +1-800-555-1234";
string phonePattern = @"\+\d{1,3}-\d{3}-\d{3}-\d{4}";

MatchCollection matches = Regex.Matches(text, phonePattern);
foreach (Match match in matches)
```

```
{
    Console.WriteLine(match.Value);
}

// Замена с использованием регулярных выражений
string dirtyText = "Цена: 1000 руб. Скидка: 200 руб.";
string cleanText = Regex.Replace(dirtyText, @"\d+", "***");
// Результат: "Цена: *** руб. Скидка: *** руб."
```

Класс Regex и его методы:

```
// Компилированное регулярное выражение (для многократного использования)
Regex emailRegex = new Regex(@"^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$");

// Проверка email
string email = "user@example.com";
bool isValidEmail = emailRegex.IsMatch(email);

// Разделение строки с помощью регулярного выражения
string complexData = "Иван,25;Мария,30;Петр,35";
string[] records = Regex.Split(complexData, @"[;,]");

// Извлечение групп
string html = "<div class='title'>Заголовок</div>";
Match match = Regex.Match(html, @"<div class='(.+?)'>(.+?)</div>");
if (match.Success)
{
    string className = match.Groups[1].Value; // "title"
    string content = match.Groups[2].Value;   // "Заголовок"
}
```

Практические примеры регулярных выражений:

```
// Валидация пароля (минимум 8 символов, буквы и цифры)
Regex passwordRegex = new Regex(@"^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{8,}$");

// Извлечение всех URL из текста
string textWithUrls = "Посетите сайты: https://example.com и http://test.ru";
Regex urlRegex = new Regex(@"https?://[^\s]+");
MatchCollection urlMatches = urlRegex.Matches(textWithUrls);

// Форматирование номера телефона
string phone = "81234567890";
string formattedPhone = Regex.Replace(phone, @"(\d{1})(\d{3})(\d{3})(\d{2})
(\d{2})",
                                      "+$1 ($2) $3-$4-$5");
// Результат: "+8 (123) 456-78-90"
```

Резюме

1. **Строки в .NET неизменяемы** - любая операция модификации создает новую строку, что важно учитывать для производительности.
2. **Разнообразие способов создания** - от простых литералов до интерполированных строк, каждый способ имеет свои преимущества.
3. **Богатый набор операций** - сравнение, поиск, извлечение подстрок предоставляют гибкость в работе с текстовыми данными.
4. **Форматирование** - мощные средства форматирования чисел, дат и сложных структур данных.
5. **StringBuilder для эффективности** - обязателен к использованию при множественных операциях модификации строк.
6. **Регулярные выражения** - незаменимый инструмент для сложного поиска и обработки текста по шаблонам.

Рекомендации по использованию:

- Используйте `string.Empty` вместо `""` для пустых строк
- Для культурно-независимого сравнения используйте `StringComparison.Ordinal`
- При множественных конкатенациях всегда используйте `StringBuilder`
- Регулярные выражения компилируйте при многократном использовании
- Используйте интерполированные строки для улучшения читаемости кода

Правильная работа со строками - фундаментальный навык в .NET разработке, напрямую влияющий на производительность и надежность приложений.