Информационные технологии и программирование

Лекция 5. Основы языка C#

Содержание лекции:

- Тип данных String (Строки)
- Основные методы
- Проблема сложения строк
- StringBuilder — «Строитель» строк
- Сравнение скорости

Преподаватель курса: Клюкин Даниил Анатольевич, ст. преподаватель каф. ПМиИТ

Тип данных String (Строки)

Что это такое?

string — это последовательность символов текста. Внутри C# строка представляет собой массив символов char, доступных только для чтения.

Основные особенности строк в .NET:

1. Они являются ссылочными типами.
2. Они неизменяемы. Однажды, создав строку, мы больше не можем ее изменить (честным способом).
3. При сравнении сравниваются значения символов в строках.

Со строками можно работать как с массивом символов, т.е:

```
string str = “Это строка”;
```

Можно обратиться к символу строки как к элементу массива:

```
Console.WriteLine(str[0]); Ответ: ‘Э’
Console.WriteLine(str[3]); Ответ: ‘ ’
```

Можно узнать длину строки с помощью свойства Length:

```
Console.WriteLine(str.Length); Ответ: 10
```

Однако, строки – неизменяемый тип данных, заменить отдельный символ на другой нельзя!

Существуют методы «замены» символа на другой, но они создают новую строку:

```
str = str.Replace(‘о’, ‘а’); Ответ: “Эта страка”
```

Основные методы

Смена регистра

Для приведения строки к верхнему и нижнему регистру используются соответственно функции **ToUpper**() и **ToLower**():

```
string hello = "Hello world!";

Console.WriteLine(hello.ToLower()); // hello world!
Console.WriteLine(hello.ToUpper()); // HELLO WORLD!
```

Еще одна группа методов позволяет узнать начинается или заканчивается ли строка на определенную подстроку. Для этого предназначены методы **StartsWith** и **EndsWith**.
Например, в массиве строк хранится список файлов, и нам надо вывести все файлы с расширением exe:

```
var files = new string[]
{
    "myapp.exe",
    "forest.jpg",
    "main.exe",
    "book.pdf",
    "river.png"
};

for (int i = 0; i < files.Length; i++)
    if (files[i].EndsWith(".exe"))
        Console.WriteLine(files[i]);
```

Поиск в строке и в массиве

С помощью метода IndexOf мы можем определить индекс первого вхождения отдельного символа или подстроки в строке:

```
string s1 = "hello world";
char ch = 'o';
int indexOfChar = s1.IndexOf(ch); // равно 4
Console.WriteLine(indexOfChar);

string substring = "wor";
int indexOfSubstring = s1.IndexOf(substring); // равно 6
Console.WriteLine(indexOfSubstring);
```

Подобным образом действует метод **LastIndexOf**, только находит индекс последнего вхождения символа или подстроки в строку.

Разделение строк

С помощью функции Split мы можем разделить строку на массив подстрок. В качестве параметра функция Split принимает массив символов или строк, которые и будут служить разделителями. Например, подсчитаем количество слов в сроке, разделив ее по пробельным символам:

```
string text = "И поэтому все так произошло";

string[] words = text.Split(new char[] { ' ' });

foreach (string s in words)
{
    Console.WriteLine(s);
}
```

Это не лучший способ разделения по пробелам, так как во входной строке у нас могло бы быть несколько подряд идущих пробелов и в итоговый массив также бы попадали пробелы.

поэтому лучше использовать другую версию метода:

```
string text = "И поэтому все так произошло";

string[] words = text.Split(new char[] { ' ' },
   StringSplitOptions.RemoveEmptyEntries);

foreach (string s in words)
{
    Console.WriteLine(s);
}
```

Второй параметр StringSplitOptions.RemoveEmptyEntries говорит, что надо удалить все пустые подстроки.

Для объединения строк также может использоваться метод **Join**:

```
string s1 = "Добрый";
string s2 = "день,";
string s3 = "дамы и господа";
string s4 = "!";
string[] values = new string[] { s1, s2, s3, s4 };

string s = string.Join(" ", values);
Console.WriteLine(s); // Добрый день, дамы и господа!
```

```
Добрый день, дамы и господа !
```

Обрезка строки

Для обрезки начальных или концевых символов используется функция **Trim**:

```
string text = " hello world ";

text = text.Trim(); // результат "hello world"
text = text.Trim(new char[] { 'd', 'h' }); //результат "ello worl“
```

Функция **Trim** без параметров обрезает начальные и конечные пробелы и возвращает обрезанную строку. Чтобы явным образом указать, какие начальные и конечные символы следует обрезать, мы можем передать в функцию массив этих символов.

Эта функция имеет частичные аналоги: функция **TrimStart** обрезает начальные символы, а функция **TrimEnd** обрезает конечные символы.

Обрезать определенную часть строки позволяет функция **Substring**:

```
string text = "Хороший день";
// обрезаем начиная с третьего символа
text = text.Substring(2);
// результат "роший день"
Console.WriteLine(text);
// обрезаем сначала до последних двух символов
text = text.Substring(0, text.Length - 2);
// результат "роший де"
Console.WriteLine(text);
```

Функция Substring также возвращает обрезанную строку. В качестве параметра первая использованная версия применяет индекс, начиная с которого надо обрезать строку. Вторая версия применяет два параметра - индекс начала обрезки и длину вырезаемой части строки.

Вставка

Для вставки одной строки в другую применяется функция **Insert**:

```
string text = "Хороший день";
string substring = "замечательный ";

text = text.Insert(8, substring);
Console.WriteLine(text);    // Хороший замечательный день
```

Первым параметром в функции Insert является индекс, по которому надо вставлять подстроку, а второй параметр - собственно подстрока.

Удаление строк

Удалить часть строки помогает метод **Remove**:

```
string text = "Хороший день";
// индекс последнего символа
int ind = text.Length - 1;
// вырезаем последний символ
text = text.Remove(ind);
Console.WriteLine(text);    // Хороший ден

// вырезаем первые два символа
text = text.Remove(0, 2);
Console.WriteLine(text);    // роший ден
```

Первая версия метода Remove принимает индекс в строке, начиная с которого надо удалить все символы. Вторая версия принимает еще один параметр - сколько символов надо удалить.

Замена

Чтобы заменить один символ или подстроку на другую, применяется метод **Replace**:

```
string text = "хороший день";

text = text.Replace("хороший", "плохой");
Console.WriteLine(text);    // плохой день

text = text.Replace("о", "");
Console.WriteLine(text);    // плхй день
```

Во втором случае применения функции Replace строка из одного символа "о" заменяется на пустую строку, то есть фактически удаляется из текста. Подобным способом легко удалять какой-то определенный текст в строках.

Сравнение

Для сравнения объектов можно применять статический метод **Compare:**

```
string s1 = "hello";
string s2 = "world";
int result = string.Compare(s1, s2);
if (result < 0)
    Console.WriteLine($"Строка {s1} стоит перед строкой {s2}");
else if (result > 0)
    Console.WriteLine($"Строка {s1} стоит после строки {s2}");
else
   Console.WriteLine($"Строки {s1} и {s2} идентичны");
// результатом будет "Строка hello стоит перед строкой world"
```

Данная версия метода **Compare** принимает две строки и возвращает число. Если первая строка по алфавиту стоит выше второй, то возвращается -1. В противном случае возвращается 1. И третий случай - если строки равны, то возвращается число 0.

В данном случае так как символ h по алфавиту стоит выше символа w, то и первая строка будет стоять выше.

Интернирование строк — это механизм, при котором одинаковые литералы представляют собой один объект в памяти.

В рамках процесса существует одна внутренняя хеш-таблица, ключами которой являются строки, а значениями – ссылки на них. Во время JIT-компиляции литеральные строки последовательно заносятся в таблицу (каждая строка в таблице встречается только один раз). На этапе выполнения ссылки на литеральные строки присваиваются из этой таблицы.

```
var s1 = "habrahabr";
var s2 = "habrahabr";
var s3 = "habra" + "habr";
Console.WriteLine(object.ReferenceEquals(s1, s2));//true
Console.WriteLine(object.ReferenceEquals(s1, s3));//true
```

Во время компиляции в IL (intermediate language) код, компилятор конкатенирует все литеральные строки, так как нет в необходимости содержать их по частям, поэтому 2-ое равенство возвращает true.

Проблема сложения строк

Формулировка проблемы:
необходимо многократно изменить строку. Например, дописывать символы в цикле.
Рассмотрим как работает конкатенация строк:

```
var str = "a" + "b" + "c";
```

Для выполнения данной инструкции будут выполнены действия:
1) Сначала выполняется сложение `"a" + "b"`, для этого происходит поиск в памяти непрерывной области для записи строки “ab”
2) Создается новая переменная для хранения “ab” в которую посимвольно переписываются левая (“a”) и правая (“a”) строки
3) Для следующего сложения “ab” и `"c"` предыдущие шаги повторяются.

Данные особенности приводят к крайне низкой скорости программы при частой перезаписи строк.

Как мы уже знаем, string — **неизменяем**. Каждое «склеивание» строк через + создает в памяти совершенно новую копию.

Пример «убийства» памяти:

```
string result = "";
for (int i = 0; i < 10000; i++)
{
    result += i.ToString(); //Создается 10,000 временных строк!
}
```

Результат: Программа тормозит, оперативная память забивается мусором, процессор занят бесконечным копированием текста.

StringBuilder — «Строитель» строк

StringBuilder — это специальный класс, который работает как динамический буфер (черновик). Он позволяет изменять текст напрямую, не создавая новые объекты в памяти.

Как это работает:

1.Вы создаете объект **StringBuilder**.

2.Добавляете в него части текста (он просто дописывает их в конец памяти).

3.В самом конце превращаете всё это в одну готовую строку.

```
using System.Text; // Нужно подключить это пространство имен!

var sb = new StringBuilder();

for (int i = 0; i < 10000; i++)
{
    sb.Append(i); // Текст дописывается в существующий буфер
}

string finalResult = sb.ToString(); // Финальная сборка одной
строки
```

Метод	Что делает
Append(value)	Добавляет текст в конец.
AppendLine(value)	Добавляет текст и переходит на новую строку.
Insert(index, value)	Вставляет текст в указанную позицию.
Replace(old, new)	Заменяет все вхождения символа или строки.
Clear()	Полностью очищает «черновик».
ToString()	Главный метод: превращает всё накопленное в обычный string.

Когда использовать?

"Правило большого пальца:

Если вы просто склеиваете 2–4 строки (например, имя и фамилию) — используйте $ (интерполяцию).

Если у вас **цикл**, который многократно что-то дописывает в строку — **всегда** берите StringBuilder. Иначе ваша программа будет «поедать» мегабайты памяти на ровном месте."

Сравнение скорости

```
var symbolsCount = 1e4;

var sb = new StringBuilder();

var sw = new Stopwatch(); // Измерение времени
sw.Start(); // Начало измерения

for (int i = 0; i < symbolsCount; i++)
{
    sb.Append("a");
}
sw.Stop(); // Конец измерения

Console.WriteLine($"1)\tЗатрачено тиков: {sw.ElapsedTicks}");
```

```
var str = "";

sw.Restart();
for (int i = 0; i < symbolsCount; i++)
{
    str += "a";
}
sw.Stop();
Console.WriteLine($"2)\tЗатрачено тиков: {sw.ElapsedTicks}");
```

```
Затрачено тиков: 802
Затрачено тиков: 279960
```

https://habr.com/ru/articles/172627/