Практическая работа №4

Строки

Цель работы: изучить работу со строками в языке C#.

Теоретическая часть

Тип string и класс String

В отличие от языков программирования, в которых строка представляет собой массив символов, в C# строки являются объектами. Следовательно, тип `string` относится к числу ссылочных.

На самом деле все строки являются экземплярами класса `String`. Имя string является алиасом для имени класса. При объявлении строк рекомендуется использовать имя `string`, а при работе с классом (например, при вызове метода) – имя `String`.

```
string str = "Строка";
char[] chararray = { 's', 't', 'r' };
string str2 = new string(chararray);
```

Рисунок 1 – Объявление строки

Содержимое объекта типа `string` не подлежит изменению. Таким образом, однажды созданную последовательность символов изменить нельзя. Если требуется строка в качестве разновидности уже имеющейся строки, то для этой цели следует создать новую строку, содержащую все необходимые изменения. Переменные ссылки на строки (т.е. объекты типа `string`) подлежат изменению, а следовательно, они могут ссылаться на другой объект. Но содержимое самого объекта типа `string` не меняется после его создания.

Следует отметить, что:

- Конкатенация строковых литералов происходит на этапе компиляции;

- Конкатенация строковых переменных происходит на этапе выполнения;
- Строки с одинаковыми значениями подвергаются интернированию и хранятся в метаданных. Это значит, что две строковых переменных ссылаются на одну и ту же строку из метаданных, хотя это два разных объекта.

В классе `System.String` предоставляется набор методов для определения длины символьных данных, поиска подстроки в текущей строке, преобразования символов из верхнего регистра в нижний и наоборот, и т.д. Обратите внимание, что этот класс является запечатанным.

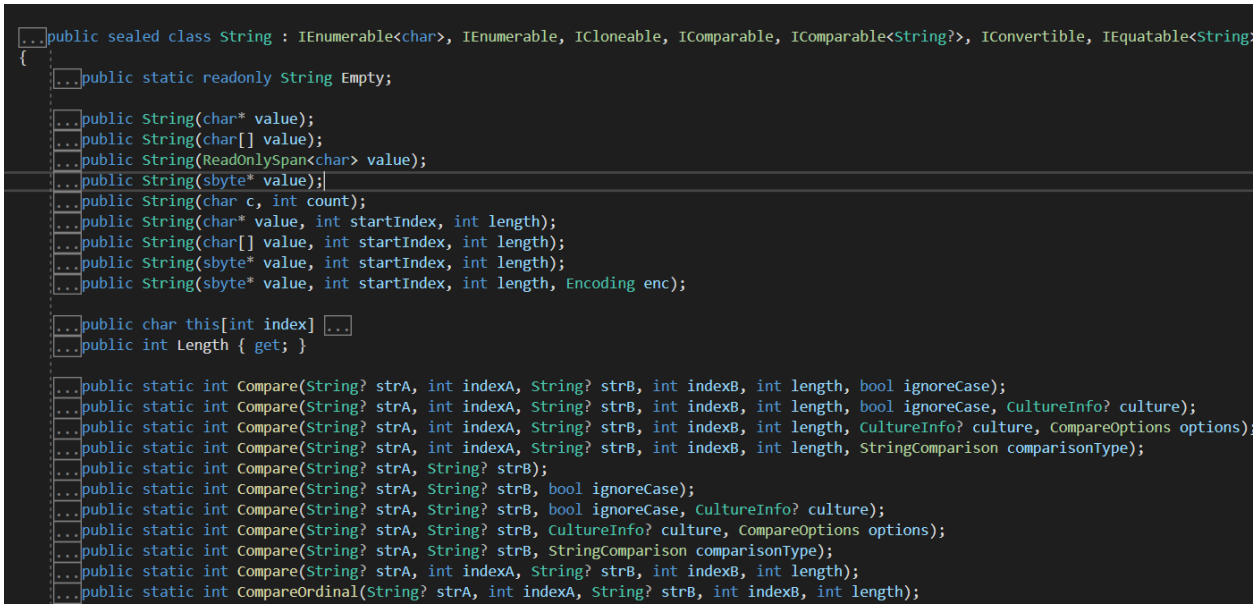

Рисунок 2 – Класс String

Рассмотрим основные члены класса:

- Поле Empty обозначает пустую строку, т.е. такую строку, которая не содержит символы. Этим оно отличается от пустой ссылки типа String, которая просто делается на несуществующий объект.
- В классе определен индексатор, который позволяет получить символ по указанному индексу. Индексация строк, как и массивов, начинается с нуля.

- Свойство Length возвращает количество символов в строке.
- Оператор == служит для проверки двух символьных строк на равенство.
- В классе есть несколько конструкторов.

```
public String(char* value);
public String(char[] value);
public String(ReadOnlySpan<char> value);
public String(sbyte* value);
public String(char c, int count);
public String(char* value, int startIndex, int length);
public String(char[] value, int startIndex, int length);
public String(sbyte* value, int startIndex, int length);
public String(sbyte* value, int startIndex, int length, Encoding enc);
```

Рисунок 3 – Конструкторы класса String

Класс StringBuilder

Хотя класс `System.String` предоставляет широкую функциональность по работе со строками, он имеет свои недостатки. Прежде всего, объект `String` представляет собой неизменяемую строку. Когда мы выполняем какой-нибудь метод класса `String`, система создает новый объект в памяти с выделением ему достаточного места. Удаление первого символа - не самая затратная операция. Но когда подобных операций множество, а объем текста, для которого надо выполнить данные операции, также не самый маленький, то издержки при потере производительности становятся более существенными.

`StringBuilder` (пространство имен `System.Text`) поможет избежать выделения строк, для этого используя свой собственный внутренний буфер символов, что обеспечит эффективное управление их последовательностями.

Он делает модификацию и конкатенацию гораздо более оптимальным образом, позволяя манипулировать символами с меньшим объемом памяти.

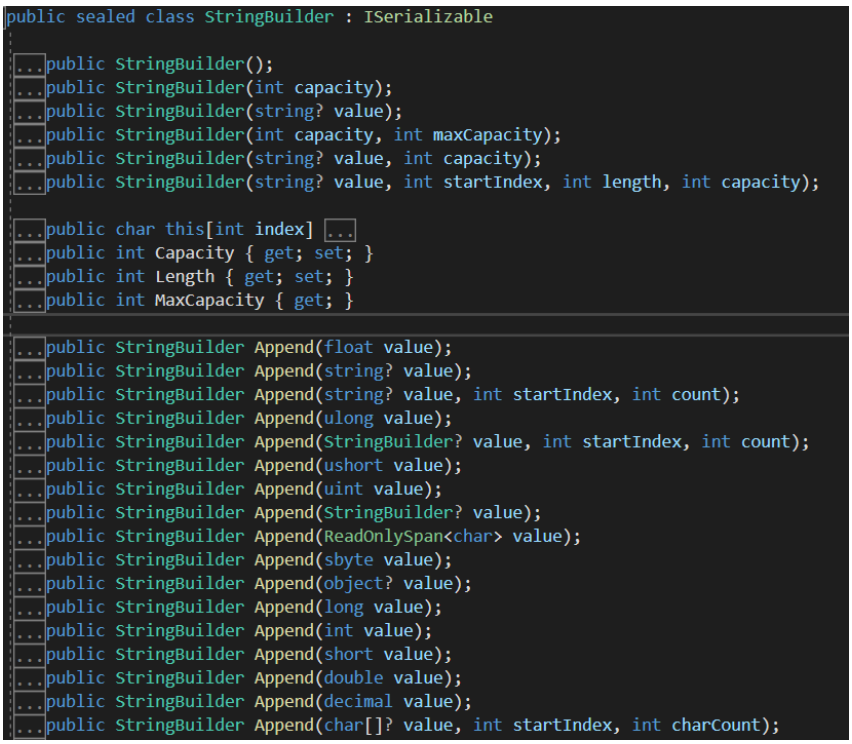

Рисунок 4 – Класс StringBuilder

Рассмотрим решение одной и той же задачи с использованием классов `String` и `StringBuilder`.

```
const string testString = "test string";
var output = string.Empty;
var iterations = int.Parse(Console.ReadLine() ?? "0");
for (var i = 0; i < iterations; i++)
{
    output += testString;
}
```

Рисунок 5 – Решение через String

```
const string testString = "test string";
var iterations = int.Parse(Console.ReadLine() ?? "0");

var str = new StringBuilder();
for (var i = 0; i < iterations; i++)
{
    str.Append(testString);
}
var output = str.ToString();
```

Рисунок 6 – Решение через SB

Такой подход не приводит к выделению новой строки на каждой итерации. Вместо этого внутренний буфер «расширяется» по мере добавления

новых символов. Это существенная экономия для тривиального изменения кода. Кроме экономии, также повышается стабильность кода. В этом и заключается смысл `StringBuilder`.

Класс `StringBuilder` рекомендуют использовать в двух случаях:

- Когда нам неизвестно количество операций и изменений над строками, которые появятся по ходу выполнения программы.
- Когда мы заведомо знаем, что приложению придется сделать множество подобных операций.

Рассмотрим основные члены класса `StringBuilder`.

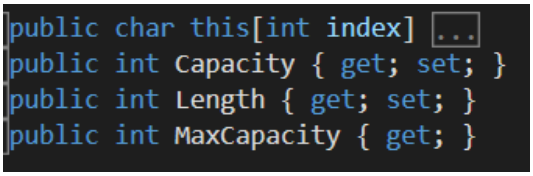

Рисунок 7 – Класс SB

- `Length` — возвращает количество символов в строке.
- `Capacity` — извлекает или задает количество символов, которое `StringBuilder` может хранить в буфере.
- `MaxCapacity` — возвращает максимальную емкость.
- `[index]` — индексатор, возвращающий символ в указанной позиции.

В классе есть несколько конструкторов, которые позволяют указать емкость (начальную и максимальную) или передать строку.

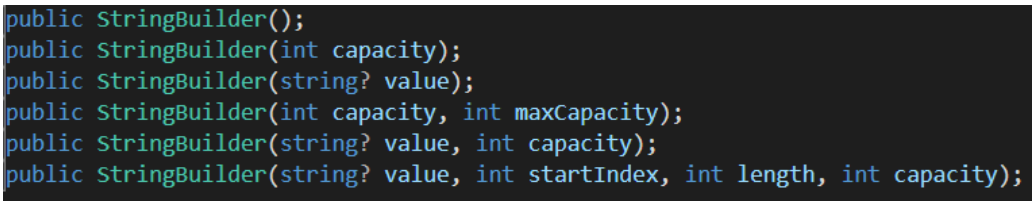

Рисунок 8 – Конструкторы класса SB

Также в классе есть методы:

- `Append` — добавляет строку после последнего символа текущей строки;
- `Insert` — вставляет подстроку в заданной позиции в текущую строку;
- `Replace` — заменяет все вхождения заданной подстроки или символа новой подстрокой или символом в текущей строке;
- `Remove` — удаляет указанные символы из текущей строки;
- `Clear` — удаляет все символы из текущей строки;
- `ToString` — преобразует значение объекта в обычную строку.

Рассмотрим пример с изменением емкости буфера.

```
StringBuilder strbuild = new StringBuilder(5);
Console.Write(strbuild.Capacity); // начальная емкость: 5
strbuild.Append("123456");
Console.Write(strbuild.Length); // 6
Console.Write(strbuild.Capacity); // вместимость удвоена: 10
Console.Write(strbuild.ToString());
```

Рисунок 9 – Пример

Практическая часть

Задания по работе с классом `String`

1) Дана строка. Написать функцию `RemoveAll`, которая удаляет из строки все вхождения заданной непустой подстроки и возвращает полученную строку в качестве результата (функция принимает два параметра — исходную строку и подстроку для удаления). Используйте `Replace`

2) Дана строка-предложение с избыточными пробелами между словами. Преобразовать ее так, чтобы между словами был ровно один пробел. Используйте `Split, string.Join`
3) Дана строка, состоящая из русских слов, набранных заглавными буквами и разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова, разделенные одним пробелом и расположенные в алфавитном порядке. Используйте `Split, Array.Sort, string.Join`

Задания по работе с классом `StringBuilder`

1) Дан символ C и строки S, S0. После каждого вхождения символа C в строку S вставить строку S0.
2) Даны две строки. Дополните более короткую пробельными символами так, чтобы их длина стала одинаковой.
3) Дана строка, изображающая десятичную запись целого положительного числа. Вывести строку, изображающую двоичную запись этого же числа.