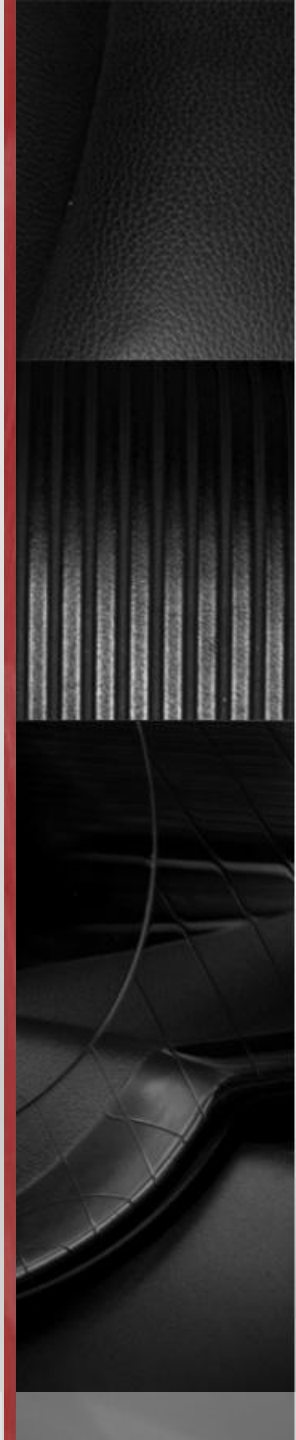
Строки

Лекция 3



Класс System.String

- System. String — класс, специально предназначенный для хранения строк и выполнения огромного числа действий с ними
- Для создания объекта этого класса (строки) в языке C# предусмотрен специальный синтаксис с использованием ключевого слова string:

```
string s;
```

- Строка в языке C# - это массив символов, предназначенный только для чтения

Доступ к отдельным символам строки

- Это означает, что можно извлекать отдельные символы из строк, применяя синтаксис индексатора массива:

```
string message = "Hello";

char char4 = message[4];   // возвращает ‘o’
```

- Однако таким же образом нельзя заменять символа в строке:

```
message[4] = ‘_’;   // ошибка!!
```

Доступ к отдельным символам строки

- Для преобразования строки путем изменения отдельных символов ее необходимо преобразовать в массив символов с помощью метода ToCharArray:

```
char[] ch = s.ToCharArray();
```

- После этого можно работать с отдельными символами как с элементами обычного массива
- Обратное преобразование массива символов в строку осуществляется путем создания нового экземпляра строки:

```
str = new string(ch);
```

Работа со строками

- В связи с исключительной важностью этого типа данных, в C# предусмотрен специальный синтаксис, который упрощает манипулирование строками
- Выполнять конкатенацию строк можно с использованием перегруженных операций:

```
string message = "Hello"; // возвращает "Hello"
message += ", Jim!"; // возвращает "Hello, Jim!"
```

Некоторые методы класса String

Название	Назначение
Compare	Статический метод. Сравнивает содержимое строк с учетом локализации
Concat	Статический метод. Объединяет несколько строк в одну строку
CopyTo	Копирует определенное число символов, начиная с определенной позиции в символьный массив с указанной позиции
IndexOf	Находит первое вхождение заданной подстроки или символа в строке
IndexOfAny	Находит первое вхождение в строку любого символа из набора
Insert	Вставляет строку в определенную позицию другой строки
LastIndexOf	То же, что IndexOf, но находит последнее вхождение
LastIndexOfAny	То же, что IndexOfAny, но находит последнее вхождение

Некоторые методы класса String

Название	Назначение
PadRight	Дополняет строку до заданной длины повторяющимся символом справа
PadLeft	Дополняет строку до заданной длины повторяющимся символом слева
Replace	Заменяет все вхождения определенного символа или подстроки другим символом или подстрокой
Split	Разбивает строку на массив подстрок, используя в качестве разделителя заданный символ
Substring	Извлекает подстроку, начиная с определенной позиции строки
ToLower	Преобразует символы строки в нижний регистр
ToUpper	Преобразует символы строки в верхний регистр
Trim	Удаляет ведущие и хвостовые символы

Неизменяемость объекта класса String

- Как уже упоминалось, String — чрезвычайно мощный класс, реализующий огромное количество очень полезных методов
- Однако с классом String связан недостаток, который делает его очень неэффективным при выполнении повторяющихся модификаций заданной строки : String является неизменяемым (immutable) типом данных

Неизменяемость объекта класса String

- Это означает, что однажды инициализированный строковый объект уже не может быть изменен
- Методы и операции, модифицирующие содержимое строк, на самом деле создают новые строки, копируя в них, при необходимости, содержимое старых строк
- Старые строки превращаются в «висящий» объект String (т.е. ни одна переменная более не указывает на него), и поэтому при следующем запуске сборщика мусора удаляется из памяти

Класс StringBuilder

- Чтобы справиться с этим, разработчики из Microsoft предусмотрели класс System.Text .StringBuilder
- Класс StringBuilder не настолько развит, как String, в смысле количества методов, которые им поддерживаются
- Обработка, которую может выполнять StringBuilder, ограничена подстановкой, добавлением и удалением текста из строк, но делает все это он гораздо более эффективно

Методы класса StringBuilder

Название	Назначение
Append()	Добавляет строку к текущей строке
Insert()	Вставляет подстроку в строку
Remove ()	Удаляет символ из текущей строки
Replace()	Заменяет все вхождения символа другим символом или вхождения подстроки другой подстрокой
ToString()	Возвращает текущую строку в виде объекта System. String (переопределение метода класса System.Object)

Объем используемой памяти

- Строки класса String используют ровно столько памяти, сколько необходимо для хранения образующих их символов, что определяется *длиной* (Length) строки
- Объем памяти, отводимой для хранения строк класса StringBuilder, в общем случае, больше минимально необходимого
- Поэтому для этих строк, кроме длины, вводят понятие *емкости* (Capacity) – выделенного объема памяти

Длина строки и емкость

- Класс StringBuilder имеет два главных свойства:
  - Length, показывающее длину строки, содержащуюся в объекте в данный момент;
  - Capacity, указывающее максимальную длину строки, которая может поместиться в выделенную для объекта память

Объем используемой памяти

- При конструировании строки классом StringBuilder первоначальное выделение памяти производится следующим образом:
  - если длина строки больше 16 символов, то выделяется объем памяти, соответствующий ее длине
  - если длина строки меньше 16 символов, то ей выделяется память для хранения 16 символов

Объем используемой памяти

- Минимальная емкость в 16 символов устанавливается по умолчанию
- Однако класс StringBuilder имеет перегруженные конструкторы, позволяющие задавать исходную емкость, а также максимальную емкость
- Отметим, что инициализировать строку StringBuilder простым присваиванием нельзя; при необходимости начальной инициализации следует использовать соответствующий конструктор

Объем используемой памяти

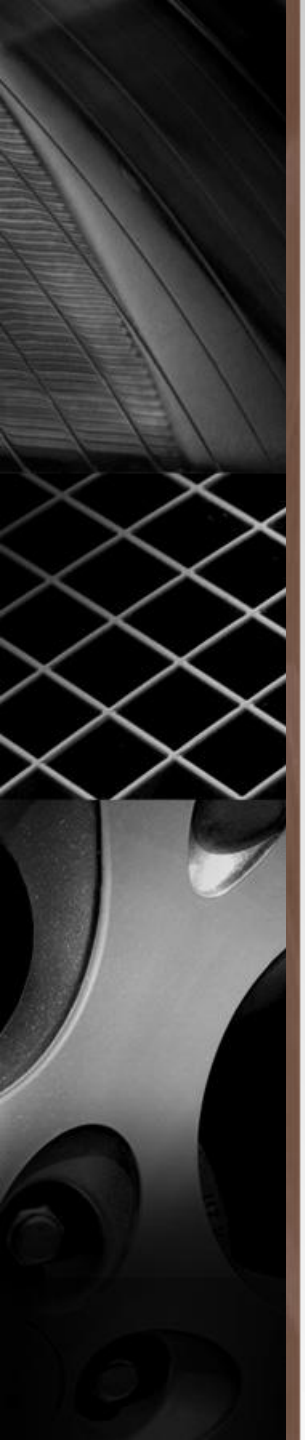
- При увеличении длины строки за пределы первоначально выделенной емкости в первый раз происходит удваивание емкости; при этом фиксируется разница между емкостью и длиной строки
- Эта разница поддерживается при всех последующих манипуляциях со строкой

Особенности StringBuilder

- Таким образом, любые модификации строки происходят внутри блока памяти, выделенного экземпляру StringBuilder
- Это делает операции добавления подстрок и замены индивидуальных символов строки очень эффективными
- После завершения всех преобразований строки StringBuilder ее обычно преобразуют в строку String с помощью переопределенного метода ToString()

Применение StringBuilder

- В основном, StringBuilder стоит применять при необходимости манипулирования многими строками
- Однако если требуется сделать что-то простое, например, соединить две строки, то лучше использовать класс String



Регулярные выражения

Определение

- Регулярным выражением называется правило обработки строк, представленное в виде так называемого *шаблона (pattern)*
- Примером такого правила являются хорошо известные шаблоны, которые используются в командах работы с файлами, например:

```
delete project?.*
```

- Этой командой из текущей папки будут удалены все файлы с наименованиями, соответствующими шаблону

Действия над строками

- С помощью регулярных выражений можно выполнять достаточно сложные и высокоуровневые действия над строками, например:
  - поиск соответствий строке шаблона;
  - извлекать данные, содержащиеся в строках;
  - преобразовать формат строк;
  - кодировать и декодировать строки

Пространство имен RegularExpression

- Технология обработки строк, основанная на использовании регулярных выражений, изначально появилась в среде UNIX и обычно используется в языке программирования Perl
- В среде .Net Framework данная технология поддерживается множеством классов из пространства имен System.Text.RegularExpressions

Шаблон регулярного выражения

- Шаблоном регулярного выражения называется специальная строка символов, задающая правило обработки обычных строк
- Шаблон рассматривается компилятором как инструкция записанная на *языке регулярных выражений*
- Компилятор преобразует шаблон в исполняемый код, непосредственно осуществляющий обработку строк

Элементы языка регулярных выражений

- Элементами языка регулярных выражений являются:
  - Escape-последовательности
  - Классы символов
  - Привязки
  - Конструкции группирования
  - Кванторы

Элементы языка регулярных выражений

- А также:
  - Конструкции обратных ссылок
  - Конструкции изменения
  - Подстановки
  - Параметры регулярных выражений
- Кроме того, регулярные выражения могут включать и обычные символы

Метасимволы

- Специальными языковыми элементами регулярных выражений являются следующие символы (метасимволы):

  `. $ ^ { [ ( | )] }* + ? \`

- Метасимвол, предваренный обратным слэш (\), рассматривается как обычный символ
- Однако сам символ \ также является метасимволом (символ начала управляющей последовательности), поэтому в регулярных выражениях должен удваиваться

Метасимволы

- Например, шаблону “1+2” будут соответствовать подстроки “12”, “112” и т.д.
- А шаблону “1\\+2” соответствует только подстрока “1+2”
- Чтобы отказаться от подобного дублирования символа \ в регулярных выражениях нужно предварять их символом @, например:

  @”1\+2”

-

Некоторые метасимволы

Метасимвол	Значение	Пример	Соответствует
^	Начало входного текста	^A	Act, April, Argument, ...
$	Конец входного текста	t$	Act, set, cat, aspect, ...
.	Любой одиночный символ, кроме перевода строки (\n)	i.ation	isation, ization, iration, ...
*	Предыдущий символ может повторяться 0 или более раз	ra*t	rt, rat, raat, raaat, ...
+	Предыдущий символ может повторяться 1 или более раз	ra+t	rat, raat, raaat, ...
?	Предыдущий символ может повторяться 0 или 1 раз	ra?t	rt, rat

Управляющие последовательности

- *Управляющие (escape) последовательности* – это последовательности одного или более символов, предваренные знаком обратного слеша (\) и имеющие специальное назначение; например, \n или \t

Некоторые escape-последовательности

Метасимвол	Значение
\f	Перевод страницы
\n	Переход на следующую строку
\t	Символ табуляции
\r	Возврат каретки
\v	Вертикальная табуляция
nnn	Символ ASCII, где *nnn* – восьмеричный код символа

Классы символов

- Класс символов соответствует какому-либо одному набору символов
- Различают следующие классы символов:
  - Положительные группы символов. Входная строка должна содержать символ из указанного набора
  - Отрицательные группы символов. Входная строка не должна содержать символ из указанного набора
  - Любой символ. Символ . (точка) в регулярных выражениях является подстановочным знаком, который соответствует всем символам, кроме \n

Классы символов

- А также:
  - Символ пробела. Входная строка может содержать любой разделитель Юникода, а также любой из множества управляющих символов
  - Символ, не являющийся пробелом. Входная строка может содержать любой символ, кроме пробела

Классы пространства RegularExpressions

- Ограничим наше рассмотрение следующими наиболее важными классами пространства имен RegularExpressions:
  - Regex
  - Match и MatchCollection
- Класс Regex непосредственно реализует механизм регулярных выражений
- Механизм отвечает за синтаксический анализ и компиляцию регулярного выражения и за выполнение заданных в нем операций

Класс Regex

- Класс Regex, обладающий богатым набором методов для обработки строк
- Статические методы Regex вызываются непосредственно без создания объекта получают в качестве одного из параметров строку шаблона

Объекты класса Regex

- Например:

```
Regex theReg = new Regex(@"(\S+)\s");
```

- Обрабатываемые строки, которые называют *входными*, передаются методам класса Regex в качестве параметров, а результатами обработки *совпадения (match)* – подстроки или коллекции подстрок исходной строки, удовлетворяющие шаблону
- Например:

```
theReg.IsMatch(“Регулярные выражения”)
```

Статические методы

- Большинство методов класса Regex имеют статические варианты и могут вызываться без создания объекта, но с указанием строки шаблона в качестве дополнительного параметра
- Например:

```
Regex.IsMatch(“Регулярные выражения”, @”(\S+)\s”);
```

- При частом изменении шаблона такой подход является предпочтительным

Методы класса Regex

Название	Описание	Число перегрузок
IsMatch()	Определяет наличие в заданной строке соответствия шаблону	4
Match()	Ищет в указанной входной строке первое вхождение заданного регулярного выражения	5
Matches()	Ищет в указанной входной строке все вхождения регулярного выражения.	4
Replace()	В указанной входной строке заменяет все строки, соответствующие шаблону регулярного выражения, указанной строкой замены	10
Split()	Разделяет входную строку в позициях, определенных шаблоном регулярного	5