Expressões regulares do .NET

Artigo • 09/03/2023

Expressões regulares oferecem um método poderoso, flexível e eficiente de processamento de texto. A extensa notação de correspondência de padrões de expressões regulares permite que você analise rapidamente grandes quantidades de texto para:

- Localizar padrões de caractere específicos.
- Validar o texto para garantir que ele corresponda a um padrão predefinido (como um endereço de email).
- Extrair, editar, substituir ou excluir substrings de texto.
- Adicionar cadeias de caracteres extraídas para uma coleção para gerar um relatório.

Para vários aplicativos que lidam com cadeias de caracteres ou que analisam grandes blocos de texto, as expressões regulares são uma ferramenta indispensável.

Como funcionam as expressões regulares

A parte mais importante do processamento de texto com expressões regulares é o mecanismo de expressão regular, que é representado pelo objeto System.Text.RegularExpressions.Regex no .NET. No mínimo, o processamento de texto usando expressões regulares exige que o mecanismo de expressões regulares seja fornecido com os dois itens informativos a seguir:

- O padrão de expressão regular a ser identificado no texto.

  No .NET, padrões de expressão regular são definidos por uma sintaxe ou linguagem especial, compatível com expressões regulares Perl 5 e inclui alguns recursos adicionais, como correspondência da direita para a esquerda. Para obter mais informações, consulte Linguagem de expressões regulares – referência rápida.

- O texto a ser analisado para o padrão de expressão regular.

Os métodos da classe Regex permitem que você realize as seguintes operações:

- Determinar se o padrão de expressão regular ocorre no texto de entrada chamando o método Regex.IsMatch. Para obter um exemplo que usa o método IsMatch para validar texto, confira Como verificar se cadeias de caracteres estão

em um formato de email válido.

- Recuperar uma ou todas as ocorrências de texto que corresponde ao padrão de expressão regular chamando o método Regex.Match ou Regex.Matches. O primeiro método retorna um objeto System.Text.RegularExpressions.Match que oferece informações sobre o texto correspondente. O último retorna um objeto MatchCollection que contém um objeto System.Text.RegularExpressions.Match para cada correspondência encontrada no texto analisado.

- Substitua o texto que corresponde ao padrão da expressão regular chamando o método Regex.Replace. Para ver exemplos que usam o método Replace para alterar formatos de data e remover caracteres inválidos de uma cadeia de caracteres, confira Como retirar caracteres inválidos de uma cadeia de caracteres e Exemplo: alterando formatos de data.

Para obter uma visão geral do modelo de objeto de expressão regular, consulte O modelo de objeto de expressão regular.

Para saber mais sobre a linguagem de expressão regular, confira Linguagem de expressão regular – referência rápida ou faça download de um dos seguintes folhetos e imprima-os:

- Referência rápida no formato Word (.docx)
- Referência rápida no formato PDF (.pdf)

Exemplos de expressões regulares

A classe String inclui métodos de pesquisa de cadeia de caracteres e substituição que podem ser usados quando você quer localizar cadeias de caracteres literais em uma cadeia de caracteres maior. Expressões regulares são mais úteis quando você quer localizar uma dentre diversas subcadeias de caracteres em uma cadeia de caracteres maior ou quando quer identificar padrões em uma cadeia de caracteres, como mostrado nos exemplos a seguir.

> ⚠ **Aviso**
>
> Ao usar **System.Text.RegularExpressions** para processar entradas não confiáveis, passe um tempo limite. Um usuário mal-intencionado pode fornecer entrada para `RegularExpressions`, causando um **ataque de negação de serviço** . APIs ASP.NET Core Framework que usam `RegularExpressions` passam um tempo limite.

> **Dica**
>
> O namespace **System.Web.RegularExpressions** contém vários objetos de expressão regular que implementam padrões predefinidos de expressão regular para analisar cadeias de caracteres de documentos HTML, XML e ASP.NET. Por exemplo, a classe **TagRegex** identifica marcas de início em uma cadeia de caracteres e a classe **CommentRegex** identifica comentários ASP.NET em uma cadeia de caracteres.

Exemplo 1: substituir substrings

Vamos presumir que uma lista de endereçamento contém nomes que, às vezes, incluem um pronome de tratamento (Sr., Srs., Srta. ou Sra.) junto com o nome e o sobrenome. Suponha que você não queira incluir os títulos ao gerar etiquetas de envelope usando a lista. Nesse caso, você pode usar uma expressão regular para remover os títulos, como o seguinte exemplo ilustra:

C#

```
using System;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      string pattern = "(Mr\\.? |Mrs\\.? |Miss |Ms\\.? )";
      string[] names = { "Mr. Henry Hunt", "Ms. Sara Samuels",
                         "Abraham Adams", "Ms. Nicole Norris" };
      foreach (string name in names)
         Console.WriteLine(Regex.Replace(name, pattern,
String.Empty));
   }
}
```

```
// The example displays the following output:
//    Henry Hunt
//    Sara Samuels
//    Abraham Adams
//    Nicole Norris
```

O padrão de expressão regular `(Mr\.? |Mrs\.? |Miss |Ms\.? )` corresponde a qualquer ocorrência de "Sr", "Sr.", "Sra", "Sra.", "Srta" ou "Srta.". A chamada para o método Regex.Replace substitui a cadeia de caracteres correspondida por String.Empty; em outras palavras, ela a remove da cadeia de caracteres original.

Exemplo 2: identificar palavras duplicadas

Duplicar palavras acidentalmente é um erro comum que escritores cometem. Use uma expressão regular para identificar palavras duplicadas, como mostrado no seguinte exemplo:

C#

```
using System;
using System.Text.RegularExpressions;

public class Class1
{
   public static void Main()
   {
      string pattern = @"\b(\w+?)\s\1\b";
      string input = "This this is a nice day. What about this?
This tastes good. I saw a a dog.";
      foreach (Match match in Regex.Matches(input, pattern,
RegexOptions.IgnoreCase))
         Console.WriteLine("{0} (duplicates '{1}') at position
{2}",
                           match.Value, match.Groups[1].Value, mat-
ch.Index);
   }
}
// The example displays the following output:
//       This this (duplicates 'This') at position 0
//       a a (duplicates 'a') at position 66
```

O padrão de expressão regular `\b(\w+?)\s\1\b` pode ser interpretado da seguinte maneira:

Padrão	Interpretação

Padrão	Interpretação
`\b`	Iniciar em um limite de palavra.
`(\w+?)`	Corresponder a um ou mais caracteres de palavra, mas o menor número de caracteres possível. Juntos, formam um grupo que pode ser chamado de `\1`.
`\s`	Corresponde a um caractere de espaço em branco.
`\1`	Corresponder à substring que é igual ao grupo denominado `\1`.
`\b`	Corresponder a um limite de palavra.

O método Regex.Matches é chamado com opções de expressão regular definidas como RegexOptions.IgnoreCase. Portanto, a operação de correspondência não faz distinção entre maiúsculas e minúsculas e o exemplo identifica a subcadeia de caracteres “This this” como uma duplicação.

A cadeia de caracteres de entrada inclui a substring "this? This”. No entanto, devido à pontuação no meio, ela não é identificada como uma duplicação.

Exemplo 3: criar dinamicamente uma expressão regular com reconhecimento de cultura

O exemplo a seguir mostra a força das expressões regulares combinada à flexibilidade oferecida pelos recursos de globalização do .NET. Ele utiliza o objeto NumberFormatInfo para determinar o formato de valores de moeda na cultura atual do sistema. Então, essa informação é usada para construir dinamicamente uma expressão regular que extrai valores de moeda do texto. Para cada correspondência, ele extrai o subgrupo que contém somente a subcadeia de caracteres, converte-a para um valor Decimal e calcula uma soma acumulada.

C#

```
using System;
using System.Collections.Generic;
using System.Globalization;
using System.Text.RegularExpressions;

public class Example
{
   public static void Main()
   {
      // Define text to be parsed.
      string input = "Office expenses on 2/13/2008:\n" +
                     "Paper (500 sheets)
```

```
$3.95\n" +
                     "Pencils (box of 10)
$1.00\n" +
                     "Pens (box of 10)
$4.49\n" +
                     "Erasers
$2.19\n" +
                     "Ink jet printer
$69.95\n\n" +
                     "Total Expenses                         $
81.58\n";

      // Get current culture's NumberFormatInfo object.
      NumberFormatInfo nfi =
CultureInfo.CurrentCulture.NumberFormat;
      // Assign needed property values to variables.
      string currencySymbol = nfi.CurrencySymbol;
      bool symbolPrecedesIfPositive = nfi.CurrencyPositivePattern %
2 == 0;
      string groupSeparator = nfi.CurrencyGroupSeparator;
      string decimalSeparator = nfi.CurrencyDecimalSeparator;

      // Form regular expression pattern.
      string pattern = Regex.Escape( symbolPrecedesIfPositive ?
currencySymbol : "") +
                       @"\s*[-+]?" + "([0-9]{0,3}(" +
groupSeparator + "[0-9]{3})*(" +
                       Regex.Escape(decimalSeparator) + "[0-9]+)?)"
+
                       (! symbolPrecedesIfPositive ? currencySymbol
: "");
      Console.WriteLine( "The regular expression pattern is:");
      Console.WriteLine("   " + pattern);

      // Get text that matches regular expression pattern.
      MatchCollection matches = Regex.Matches(input, pattern,

RegexOptions.IgnorePatternWhitespace);
      Console.WriteLine("Found {0} matches.", matches.Count);

      // Get numeric string, convert it to a value, and add it to
List object.
      List<decimal> expenses = new List<Decimal>();

      foreach (Match match in matches)
         expenses.Add(Decimal.Parse(match.Groups[1].Value));

      // Determine whether total is present and if present, whether
it is correct.
      decimal total = 0;
      foreach (decimal value in expenses)
         total += value;

      if (total / 2 == expenses[expenses.Count - 1])
```

```
         Console.WriteLine("The expenses total {0:C2}.", expen-
ses[expenses.Count - 1]);
      else
         Console.WriteLine("The expenses total {0:C2}.", total);
   }
}
// The example displays the following output:
//       The regular expression pattern is:
//          \$\s*[-+]?([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)
//       Found 6 matches.
//       The expenses total $81.58.
```

Em um computador cuja cultura atual seja Inglês – Estados Unidos (en-US), o exemplo compila dinamicamente a expressão regular `\$\s*[-+]?([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)`. Esse padrão de expressão regular pode ser interpretado da seguinte maneira:

Padrão	Interpretação
`\$`	Procure uma única ocorrência do símbolo de cifrão (`$`) na cadeia de caracteres de entrada. A cadeia de caracteres do padrão de expressão regular inclui uma barra invertida para indicar que o símbolo de cifrão deve ser interpretado literalmente ao invés de como uma âncora de expressão regular. O símbolo `$` sozinho indicaria que o mecanismo de expressão regular deveria tentar iniciar a correspondência no final de uma cadeia de caracteres. Para garantir que o símbolo de moeda da cultura atual não seja interpretado incorretamente como um símbolo de expressão regular, o exemplo chama o método Regex.Escape para escapar o caractere.
`\s*`	Procure zero ou mais ocorrências de um caractere de espaço em branco.
`[-+]?`	Procure zero ou uma ocorrência de um sinal de positivo ou negativo.
`([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)`	Os parênteses externos definem a expressão como um grupo de captura ou uma subexpressão. Se uma correspondência for localizada, informações sobre essa parte da cadeia de caracteres correspondente podem ser recuperadas do segundo objeto Group no objeto GroupCollection retornado pela propriedade Match.Groups. O primeiro elemento na coleção representa a correspondência inteira.
`[0-9]{0,3}`	Procure de zero a três ocorrências dos dígitos decimais de 0 a 9.
`(,[0-9]{3})*`	Procure zero ou mais ocorrências de um separador de grupo seguido por três dígitos decimais.
`\.`	Procure uma única ocorrência do separador decimal.

Padrão	Interpretação
`[0-9]+`	Procure por um ou mais dígitos decimais.
`(\.[0-9]+)?`	Procure zero ou uma ocorrência de um separador decimal seguido por pelo menos um dígito decimal.

Se cada subpadrão for encontrado na cadeia de caracteres de entrada, a correspondência será bem-sucedida, e um objeto Match que contém informações sobre a correspondência será adicionado ao objeto MatchCollection.

Artigos relacionados

Título	Descrição
Linguagem de expressões regulares – referência rápida	Oferece informações sobre o conjunto de caracteres, operadores e constructos que você pode usar para definir expressões regulares.
O modelo de objeto de expressão regular	Oferece informações e exemplos de código que ilustram como usar as classes de expressão regular.
Detalhes do comportamento de expressões regulares	Oferece informações sobre os recursos e o comportamento das expressões regulares do .NET.
Usar expressões regulares no Visual Studio	

Referência

- System.Text.RegularExpressions
- System.Text.RegularExpressions.Regex
- Expressões regulares - referência rápida (fazer download no formato Word)
- Expressões regulares - referência rápida (fazer download no formato PDF)

more information, see our contributor guide.

Provide product feedback