Expressions régulières du .NET Framework

.NET Framework 4.5 Ce sujet n'a pas encore été évalué

Les expressions régulières constituent une méthode puissante, souple et efficace de traitement du texte. L'utilisation intensive des caractères génériques assurée par les expressions régulières vous permet : d'analyser rapidement des volumes importants de texte pour rechercher des séquences de caractères spécifiques, de vérifier qu'un texte correspond à un modèle prédéfini (une adresse de messagerie, par exemple), d'extraire, de modifier, de remplacer ou de supprimer des sous-chaînes de texte, et d'ajouter les chaînes extraites à une collection afin de générer un rapport. Pour de nombreuses applications qui gèrent des chaînes ou analysent des blocs de texte volumineux, les expressions régulières sont un outil indispensable.

Fonctionnement des expressions régulières

La pièce maîtresse du traitement de texte avec des expressions régulières est le moteur des expressions régulières, représenté par l'objet System.Text.RegularExpressions.Regex dans le .NET Framework. Le traitement de texte à l'aide d'expressions régulières requiert au minimum que le moteur des expressions régulières soit fourni avec les deux éléments d'information suivants :

- Modèle d'expression régulière à identifier dans le texte.

  Dans le .NET Framework, les modèles d'expressions régulières sont définis au moyen d'une syntaxe spéciale ou d'un langage spécial, qui est compatible avec les expressions régulières Perl 5 et ajoute des fonctionnalités supplémentaires, telles que la recherche de correspondances de droite à gauche. Pour plus d'informations, consultez Langage des expressions régulières - Aide-mémoire.

- Texte à analyser en fonction du modèle d'expression régulière.

Les méthodes de la classe Regex vous permettent d'exécuter les opérations suivantes :

- Déterminer si le modèle d'expression régulière se trouve dans le texte d'entrée en appelant la méthode IsMatch. Pour obtenir un exemple qui utilise la méthode IsMatch afin de valider du texte, consultez Comment : vérifier que des chaînes sont dans un format d'adresse de messagerie valide.

- Récupérer une ou toutes les occurrences du texte qui correspondent au modèle d'expression régulière en appelant la méthode Match ou Matches. La première méthode retourne un objet Match qui fournit des informations sur le texte correspondant. La deuxième retourne un objet MatchCollection qui contient un objet Match pour chaque correspondance trouvée dans le texte analysé.

- Remplacer le texte qui correspond au modèle d'expression régulière en appelant la méthode Replace. Pour obtenir des exemples qui utilisent la méthode Replace afin de modifier les formats de date et supprimer des caractères non valides d'une chaîne, consultez Comment : supprimer des caractères non valides d'une chaîne et Exemple : modification des formats de date.

Pour une vue d'ensemble du modèle objet d'expression régulière, consultez Modèle objet d'expression régulière.

Exemples d'expressions régulières

La classe String inclut plusieurs méthodes recherche et de remplacement de chaînes que vous pouvez utiliser lorsque vous voulez trouver des chaînes littérales dans une chaîne plus grande. Les expressions régulières sont surtout utiles lorsque

vous voulez trouver une des nombreuses sous-chaînes d'une chaîne plus grande, ou lorsque vous voulez identifier des modèles dans une chaîne, comme illustré par les exemples suivants.

Exemple 1: remplacement de sous-chaînes

Supposez qu'une liste de diffusion contienne des noms qui incluent parfois un titre (M., Mme, Mlle ou Mlle) avec le prénom et le nom. Si vous ne voulez pas inclure les titres lorsque vous générez des étiquettes d'enveloppe à partir de la liste, vous pouvez utiliser une expression régulière pour supprimer les titres, comme illustré par l'exemple suivant.

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
```

Le modèle d'expression régulière `(Mr\.? |Mrs\.? |Miss |Ms\.? )` correspond à toutes les occurrences de « Mr », « Mr. » " , « Mrs », « Mrs. » " , « Miss », « Ms » ou « Ms. ». " . L'appel à la méthode Regex.Replace remplace la chaîne correspondante par String.Empty ; en d'autres termes, il la supprime de la chaîne d'origine.

Exemple 2: identification de mots dupliqués

La duplication accidentelle de mots est une erreur courante que font les rédacteurs. Il est possible d'utiliser une expression régulière pour identifier des mots dupliqués, comme illustré par l'exemple suivant.

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
      foreach (Match match in Regex.Matches(input, pattern, RegexOptions.IgnoreCase))
         Console.WriteLine("{0} (duplicates '{1}') at position {2}",
                           match.Value, match.Groups[1].Value, match.Index);
   }
}
// The example displays the following output:
//       This this (duplicates 'This)' at position 0
//       a a (duplicates 'a)' at position 66
```

Le modèle d'expression régulière `\b(\w+?)\s\1\b` peut être interprété comme suit :

`\b`	Commencer à la limite d'un mot.
(\w+)	Mettre en correspondance un ou plusieurs caractères alphabétiques. Ensemble, ils forment un groupe qui peut être désigné en tant que `\1`.
`\s`	Mettre en correspondance un espace blanc.
`\1`	Mettre en correspondance la sous-chaîne qui est égale au groupe nommé `\1`.
`\b`	Mettre en correspondance la limite d'un mot.

La méthode Regex.Matches est appelée avec RegexOptions.IgnoreCase comme valeur pour les options d'expression régulière. Par conséquent, l'opération de correspondance ne respecte pas la casse, et l'exemple identifie la sous-chaîne "This this" comme une duplication.

Notez que la chaîne d'entrée inclut la sous-chaîne « this ? This". Toutefois, en raison du signe de ponctuation intermédiaire, elle n'est pas identifiée comme une duplication.

Exemple 3: génération dynamique d'une expression régulière dépendante de la culture

L'exemple suivant illustre la puissance des expressions régulières alliée à la souplesse qu'offrent les fonctionnalités de globalisation du .NET Framework. Il utilise l'objet NumberFormatInfo pour déterminer le format de valeurs monétaires dans la culture actuelle du système. Il utilise ensuite cette information pour construire de façon dynamique une expression régulière capable d'extraire les valeurs monétaires du texte. Pour chaque correspondance, il extrait le sous-groupe qui contient uniquement la chaîne numérique, qu'il convertit en valeur Decimal avant de calculer un total cumulé.

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}.", expenses[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.
```

Sur un ordinateur dont la culture actuelle est Anglais - États-Unis (en-US), l'exemple génère l'expression régulière `\$\s*[-+]?([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)` de façon dynamique. Ce modèle d'expression régulière peut être interprété comme suit :

\$	Rechercher une seule occurrence du symbole dollar ($) dans la chaîne d'entrée. La chaîne du modèle d'expression régulière comprend une barre oblique inverse pour indiquer que le symbole dollar doit être interprété littéralement, et non comme une ancre d'expression régulière. (Le symbole $ seul indiquerait que le moteur des expressions régulières doit essayer de commencer sa correspondance à la fin d'une chaîne.) Pour s'assurer que le symbole monétaire de la culture actuelle ne sera pas interprété à tort comme un symbole d'expression régulière, l'exemple appelle la méthode Escape afin de créer une séquence d'échappement pour le caractère.
\s*	Rechercher zéro, une ou plusieurs occurrences d'un espace blanc.
[-+]?	Rechercher zéro ou une occurrence d'un signe positif ou d'un signe négatif.
([0-9]{0,3}(,[0-9]{3})*(\.[0-9]+)?)	Les parenthèses externes qui encadrent cette expression la définissent en tant que groupe de capture ou sous-expression. Si une correspondance est trouvée, les informations relatives à cette partie de la chaîne correspondante peuvent être récupérées à partir du deuxième objet Group de l'objet GroupCollection retourné par la propriété Match.Groups. (Le premier élément de la collection représente l'intégralité de la correspondance.)
[0-9]{0,3}	Rechercher de zéro à trois occurrences des chiffres décimaux 0 à 9.
(,[0-9]{3})*	Rechercher zéro, une ou plusieurs occurrences d'un séparateur de groupes suivi de trois chiffres décimaux.

\.	Rechercher une seule occurrence du séparateur décimal.
[0-9]+	Rechercher un ou plusieurs chiffres décimaux.
(\.[0-9]+)?	Rechercher zéro ou une occurrence du séparateur décimal suivie d'au moins un chiffre décimal.

Si chacun de ces sous-modèles est trouvé dans la chaîne d'entrée, la mise en correspondance aboutit et un objet Match qui contient des informations sur la correspondance est ajouté à l'objet MatchCollection.

Rubriques connexes

Titre	Description
Langage des expressions régulières - Aide-mémoire	Fournit des informations sur l'ensemble des caractères, opérateurs et constructions que vous pouvez utiliser pour définir des expressions régulières.
Modèle objet d'expression régulière	Fournit des informations et des exemples de code qui montrent comment utiliser les classes d'expressions régulières.
Comportement détaillé des expressions régulières	Fournit des informations sur les capacités et le comportement des expressions régulières du .NET Framework.
Exemples d'expressions régulières	Fournit des exemples de code qui montrent des utilisations courantes d'expressions régulières.

Référence

System.Text.RegularExpressions

System.Text.RegularExpressions.Regex

Ajouts de la communauté