C# 预处理器指令 C# 异常处理

# C# 正则表达式

**正则表达式** 是一种匹配输入文本的模式。.Net 框架提供了允许这种匹配的正则表达式引擎。模式由一个或多个字符、运算符和结构组成。

## 定义正则表达式

下面列出了用于定义正则表达式的各种类别的字符、运算符和结构。

- 字符转义
- 字符类
- 定位点
- 分组构造
- 限定符
- 反向引用构造
- 备用构造
- 替换
- 杂项构造

### 字符转义

正则表达式中的反斜杠字符（\）指示其后跟的字符是特殊字符，或应按原义解释该字符。

下表列出了转义字符：

| 转义字符 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **\a** | 与报警 (bell) 符 \u0007 匹配。 | \a | "Warning!" + '\u0007' 中的 "\u0007" |
| **\b** | 在字符类中，与退格键 \u0008 匹配。 | [\b]{3,} | "\b\b\b\b" 中的 "\b\b\b\b" |
| **\t** | 与制表符 \u0009 匹配。 | (\w+)\t | "Name\tAddr\t" 中的 "Name\t" 和 "Addr\t" |
| **\r** | 与回车符 \u000D 匹配。（\r 与换行符 \n 不是等效的。） | \r\n(\w+) | "\r\Hello\nWorld." 中的 "\r\nHello" |
| **\v** | 与垂直制表符 \u000B 匹配。 | [\v]{2,} | "\v\v\v" 中的 "\v\v\v" |
| **\f** | 与换页符 \u000C 匹配。 | [\f]{2,} | "\f\f\f" 中的 "\f\f\f" |
| **\n** | 与换行符 \u000A 匹配。 | \r\n(\w+) | "\r\Hello\nWorld." 中的 "\r\nHello" |

| | | | |
|---|---|---|---|
| **\e** | 与转义符 \u001B 匹配。 | \e | "\x001B" 中的 "\x001B" |
| **\ nnn** | 使用八进制表示形式指定一个字符（nnn 由二到三位数字组成）。 | \w\040\w | "a bc d" 中的 "a b" 和 "c d" |
| **\x nn** | 使用十六进制表示形式指定字符（nn 恰好由两位数字组成）。 | \w\x20\w | "a bc d" 中的 "a b" 和 "c d" |
| **\c X \c x** | 匹配 X 或 x 指定的 ASCII 控件字符，其中 X 或 x 是控件字符的字母。 | \cC | "\x0003" 中的 "\x0003" (Ctrl-C) |
| **\u nnnn** | 使用十六进制表示形式匹配一个 Unicode 字符（由 nnnn 表示的四位数）。 | \w\u0020\w | "a bc d" 中的 "a b" 和 "c d" |
| \ | 在后面带有不识别的转义字符时，与该字符匹配。 | \d+[\+-x*]\d+\d+[\+-x*\d+ | "(2+2) * 3*9" 中的 "2+2" 和 "3*9" |

## 字符类

字符类与一组字符中的任何一个字符匹配。

下表列出了字符类：

| 字符类 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **[character_group]** | 匹配 character_group 中的任何单个字符。默认情况下，匹配区分大小写。 | [mn] | "mat" 中的 "m"，"moon" 中的 "m" 和 "n" |
| **[^character_group]** | 非：与不在 character_group 中的任何单个字符匹配。 默认情况下，character_group 中的字符区分大小写。 | [^aei] | "avail" 中的 "v" 和 "l" |
| **[ first - last ]** | 字符范围：与从 first 到 last 的范围中的任何单个字符匹配。 | [b-d] | [b-d]irds 可以匹配 Birds、Cirds、 Dirds |
| **.** | 通配符：与除 \n 之外的任何单个字符匹配。<br>若要匹配原意句点字符（. 或 \u002E），您必须在该字符前面加上转义符 (\.)。 | a.e | "have" 中的 "ave"， "mate" 中的 "ate" |
| **\p{ name }** | 与 *name* 指定的 Unicode 通用类别或命名块中的任何单个字符匹配。 | \p{Lu} | "City Lights" 中的 "C" 和 "L" |
| **\P{ name }** | 与不在 *name* 指定的 Unicode 通用类别或命名块中的任何单个字符匹配。 | \P{Lu} | "City" 中的 "i"、 "t" 和 "y" |

| | | | |
|---|---|---|---|
| **\w** | 与任何单词字符匹配。 | \w | "Room#1" 中的 "R"、 "o"、 "m" 和 "1" |
| **\W** | 与任何非单词字符匹配。 | \W | "Room#1" 中的 "#" |
| **\s** | 与任何空白字符匹配。 | \w\s | "ID A1.3" 中的 "D " |
| **\S** | 与任何非空白字符匹配。 | \s\S | "int __ctr" 中的 " _" |
| **\d** | 与任何十进制数字匹配。 | \d | "4 = IV" 中的 "4" |
| **\D** | 匹配不是十进制数的任意字符。 | \D | "4 = IV" 中的 " "、 "="、 " "、 "I" 和 "V" |

## 定位点

定位点或原子零宽度断言会使匹配成功或失败，具体取决于字符串中的当前位置，但它们不会使引擎在字符串中前进或使用字符。

下表列出了定位点：

| 断言 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **^** | 匹配必须从字符串或一行的开头开始。 | ^\d{3} | "567-777-" 中的 "567" |
| **$** | 匹配必须出现在字符串的末尾或出现在行或字符串末尾的 **\n** 之前。 | -\d{4}$ | "8-12-2012" 中的 "-2012" |
| **\A** | 匹配必须出现在字符串的开头。 | \A\w{4} | "Code-007-" 中的 "Code" |
| **\Z** | 匹配必须出现在字符串的末尾或出现在字符串末尾的 **\n** 之前。 | -\d{3}\Z | "Bond-901-007" 中的 "-007" |
| **\z** | 匹配必须出现在字符串的末尾。 | -\d{3}\z | "-901-333" 中的 "-333" |
| **\G** | 匹配必须出现在上一个匹配结束的地方。 | \G\(\d\) | "(1)(3)(5)[7](9)" 中的 "(1)"、 "(3)" 和 "(5)" |
| **\b** | 匹配一个单词边界，也就是指单词和空格间的位置。 | er\b | 匹配"never"中的"er"，但不能匹配"verb"中的"er"。 |
| **\B** | 匹配非单词边界。 | er\B | 匹配"verb"中的"er"，但不能匹配"never"中的"er"。 |

## 分组构造

分组构造描述了正则表达式的子表达式，通常用于捕获输入字符串的子字符串。

下表列出了分组构造：

| 分组构造 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **( subexpression )** | 捕获匹配的子表达式并将其分配到一个从零开始的序号中。 | (\w)\1 | "deep" 中的 "ee" |
| **(?< name >subexpression)** | 将匹配的子表达式捕获到一个命名组中。 | (?< double>\w)\k< double> | "deep" 中的 "ee" |
| **(?< name1 -name2 >subexpression)** | 定义平衡组定义。 | (((?'Open'\()[^\(\)]*)+((?'Close-Open'\))[^\(\)]*)+)*(?(Open)(?!))$ | "3+2^((1-3)*(3-1))" 中的 "((1-3)*(3-1))" |
| **(?: subexpression)** | 定义非捕获组。 | Write(?:Line)? | "Console.WriteLine()" 中的 "WriteLine" |
| **(?imnsx-imnsx:subexpression)** | 应用或禁用 *subexpression* 中指定的选项。 | A\d{2}(?i:\w+)\b | "A12xl A12XL a12xl" 中的 "A12xl" 和 "A12XL" |
| **(?= subexpression)** | 零宽度正预测先行断言。 | \w+(?=\.) | "He is. The dog ran. The sun is out." 中的 "is"、 "ran" 和 "out" |
| **(?! subexpression)** | 零宽度负预测先行断言。 | \b(?!un)\w+\b | "unsure sure unity used" 中的 "sure" 和 "used" |
| **(?<=subexpression)** | 零宽度正回顾后发断言。 | (?<=19)\d{2}\b | "1851 1999 1950 1905 2003" 中的 "99"、"50"和 "05" |
| **(?<! subexpression)** | 零宽度负回顾后发断言。 | (? | "Hi woman Hi man" 中的 "man" |
| **(?> subexpression)** | 非回溯（也称为"贪婪"）子表达式。 | [13579](?>A+B+) | "1ABB 3ABBC 5AB 5AC" 中的 "1ABB"、 "3ABB" 和 "5AB" |

**实例**

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

public class Example
{
   public static void Main()
   {
      string input = "1851 1999 1950 1905 2003";
      string pattern = @"(?<=19)\d{2}\b";

      foreach (Match match in Regex.Matches(input, pattern))
```

```
            Console.WriteLine(match.Value);
        }
    }
```

运行实例 »

## 限定符

限定符指定在输入字符串中必须存在上一个元素（可以是字符、组或字符类）的多少个实例才能出现匹配项。 限定符包括下表中列出的语言元素。

下表列出了限定符：

| 限定符 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| * | 匹配上一个元素零次或多次。 | \d*\.\d | ".0"、 "19.9"、 "219.9" |
| + | 匹配上一个元素一次或多次。 | "be+" | "been" 中的 "bee"， "bent" 中的 "be" |
| ? | 匹配上一个元素零次或一次。 | "rai?n" | "ran"、 "rain" |
| { n } | 匹配上一个元素恰好 n 次。 | ",\d{3}" | "1,043.6" 中的 ",043"，"9,876,543,210" 中的 ",876"、",543" 和 ",210" |
| { n ,} | 匹配上一个元素至少 n 次。 | "\d{2,}" | "166"、 "29"、 "1930" |
| { n , m } | 匹配上一个元素至少 n 次，但不多于 m 次。 | "\d{3,5}" | "166"， "17668"， "193024" 中的 "19302" |
| *? | 匹配上一个元素零次或多次，但次数尽可能少。 | \d*?\.\d | ".0"、 "19.9"、 "219.9" |
| +? | 匹配上一个元素一次或多次，但次数尽可能少。 | "be+?" | "been" 中的 "be"， "bent" 中的 "be" |
| ?? | 匹配上一个元素零次或一次，但次数尽可能少。 | "rai??n" | "ran"、 "rain" |
| { n }? | 匹配前导元素恰好 n 次。 | ",\d{3}?" | "1,043.6" 中的 ",043"，"9,876,543,210" 中的 ",876"、",543" 和 ",210" |
| { n ,}? | 匹配上一个元素至少 n 次，但次数尽可能少。 | "\d{2,}?" | "166"、 "29" 和 "1930" |
| { n , m }? | 匹配上一个元素的次数介于 n 和 m 之间，但次数尽可能少。 | "\d{3,5}?" | "166"， "17668"， "193024" 中的 "193" 和 "024" |

## 反向引用构造

反向引用允许在同一正则表达式中随后标识以前匹配的子表达式。

下表列出了反向引用构造：

| 反向引用构造 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **\ number** | 反向引用。 匹配编号子表达式的值。 | (\w)\1 | "seek" 中的 "ee" |
| **\k< name >** | 命名反向引用。 匹配命名表达式的值。 | (?< char>\w)\k< char> | "seek" 中的 "ee" |

## 备用构造

备用构造用于修改正则表达式以启用 either/or 匹配。

下表列出了备用构造：

| 备用构造 | 描述 | 模式 | 匹配 |
|---|---|---|---|
| **\|** | 匹配以竖线 (\|) 字符分隔的任何一个元素。 | th(e\|is\|at) | "this is the day. " 中的 "the" 和 "this" |
| **(?( expression )yes \| no )** | 如果正则表达式模式由 expression 匹配指定，则匹配 *yes*；否则匹配可选的 *no* 部分。 expression 被解释为零宽度断言。 | (?(A)A\d{2}\b\|\b\d{3}\b) | "A10 C103 910" 中的 "A10" 和 "910" |
| **(?( name )yes \| no )** | 如果 name 或已命名或已编号的捕获组具有匹配，则匹配 *yes*；否则匹配可选的 *no*。 | (?< quoted>")?(?(quoted).+?"\|\S+\s) | "Dogs.jpg "Yiska playing.jpg"" 中的 Dogs.jpg 和 "Yiska playing.jpg" |

## 替换

替换是替换模式中使用的正则表达式。

下表列出了用于替换的字符：

| 字符 | 描述 | 模式 | 替换模式 | 输入字符串 | 结果字符串 |
|---|---|---|---|---|---|
| **$**number | 替换按组 *number* 匹配的子字符串。 | \b(\w+)(\s)(\w+)\b | $3$2$1 | "one two" | "two one" |
| **${**name**}** | 替换按命名组 *name* 匹配的子字符串。 | \b(?< word1>\w+)(\s)(?< word2>\w+)\b | ${word2} ${word1} | "one two" | "two one" |
| **$$** | 替换字符"$"。 | \b(\d+)\s?USD | $$$1 | "103 USD" | "$103" |
| **$&** | 替换整个匹配项的一个副本。 | (\$*(\d*(\.+\d+)?){1}) | **$& | "$1.30" | "**$1.30**" |
| **$`** | 替换匹配前的输入字符串的所有文本。 | B+ | $` | "AABBCC" | "AAAACC" |

| | | | | | |
|---|---|---|---|---|---|
| **$'** | 替换匹配后的输入字符串的所有文本。 | B+ | $' | "AABBCC" | "AACCCC" |
| **$+** | 替换最后捕获的组。 | B+(C+) | $+ | "AABBCCDD" | AACCDD |
| **$_** | 替换整个输入字符串。 | B+ | $_ | "AABBCC" | "AAAABBCCCC" |

### 杂项构造

下表列出了各种杂项构造：

| 构造 | 描述 | 实例 |
|---|---|---|
| **(?imnsx-imnsx)** | 在模式中间对诸如不区分大小写这样的选项进行设置或禁用。 | \bA(?i)b\w+\b 匹配 "ABA Able Act" 中的 "ABA" 和 "Able" |
| **(?#注释)** | 内联注释。该注释在第一个右括号处终止。 | \bA(?#匹配以A开头的单词)\w+\b |
| **#** [行尾] | 该注释以非转义的 # 开头，并继续到行的结尾。 | (?x)\bA\w+\b#匹配以 A 开头的单词 |

## Regex 类

Regex 类用于表示一个正则表达式。

下表列出了 Regex 类中一些常用的方法：

| 序号 | 方法 & 描述 |
|---|---|
| 1 | **public bool IsMatch( string input )**<br>指示 Regex 构造函数中指定的正则表达式是否在指定的输入字符串中找到匹配项。 |
| 2 | **public bool IsMatch( string input, int startat )**<br>指示 Regex 构造函数中指定的正则表达式是否在指定的输入字符串中找到匹配项，从字符串中指定的开始位置开始。 |
| 3 | **public static bool IsMatch( string input, string pattern )**<br>指示指定的正则表达式是否在指定的输入字符串中找到匹配项。 |
| 4 | **public MatchCollection Matches( string input )**<br>在指定的输入字符串中搜索正则表达式的所有匹配项。 |
| 5 | **public string Replace( string input, string replacement )**<br>在指定的输入字符串中，把所有匹配正则表达式模式的所有匹配的字符串替换为指定的替换字符串。 |
| 6 | **public string[] Split( string input )**<br>把输入字符串分割为子字符串数组，根据在 Regex 构造函数中指定的正则表达式模式定义的位置进行分割。 |

如需了解 Regex 类的完整的属性列表，请参阅微软的 C# 文档。

# 实例 1

下面的实例匹配了以 'S' 开头的单词：

**实例**

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

namespace RegExApplication
{
   class Program
   {
      private static void showMatch(string text, string expr)
      {
         Console.WriteLine("The Expression: " + expr);
         MatchCollection mc = Regex.Matches(text, expr);
         foreach (Match m in mc)
         {
            Console.WriteLine(m);
         }
      }
      static void Main(string[] args)
      {
         string str = "A Thousand Splendid Suns";

         Console.WriteLine("Matching words that start with 'S': ");
         showMatch(str, @"\bS\S*");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Matching words that start with 'S':
The Expression: \bS\S*
Splendid
Suns
```

# 实例 2

下面的实例匹配了以 'm' 开头以 'e' 结尾的单词：

**实例**

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

namespace RegExApplication
{
   class Program
```

```
   {
      private static void showMatch(string text, string expr)
      {
         Console.WriteLine("The Expression: " + expr);
         MatchCollection mc = Regex.Matches(text, expr);
         foreach (Match m in mc)
         {
            Console.WriteLine(m);
         }
      }
      static void Main(string[] args)
      {
         string str = "make maze and manage to measure it";

         Console.WriteLine("Matching words start with 'm' and ends with 'e':");
         showMatch(str, @"\bm\S*e\b");
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Matching words start with 'm' and ends with 'e':
The Expression: \bm\S*e\b
make
maze
manage
measure
```

## 实例 3

下面的实例替换掉多余的空格：

**实例**

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

namespace RegExApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         string input = "Hello   World   ";
         string pattern = "\\s+";
         string replacement = " ";
         Regex rgx = new Regex(pattern);
         string result = rgx.Replace(input, replacement);

         Console.WriteLine("Original String: {0}", input);
```

```
            Console.WriteLine("Replacement String: {0}", result);
            Console.ReadKey();
        }
    }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
Original String: Hello   World
Replacement String: Hello World
```

C# 预处理器指令 C# 异常处理

点我分享笔记