1. C#正则表达式符号模式

|  |  |
| --- | --- |
| 字　　符 | 描　　述 |
| \ | 转义字符，将一个具有特殊功能的字符转义为一个普通字符，或反过来 |
| ^ | 匹配输入字符串的开始位置 |
| $ | 匹配输入字符串的结束位置 |
| \* | 匹配前面的零次或多次的子表达式 |
| + | 匹配前面的一次或多次的子表达式 |
| ? | 匹配前面的零次或一次的子表达式 |
| {*n*} | *n*是一个非负整数，匹配前面的*n*次子表达式 |
| {*n*,} | *n*是一个非负整数，至少匹配前面的*n*次子表达式 |
| {*n*,*m*} | *m*和*n*均为非负整数，其中*n*<=*m*，最少匹配*n*次且最多匹配*m*次 |
| ? | 当该字符紧跟在其他限制符（\*，+，?，{*n*}，{*n*,}，{*n*，*m*}）后面时，匹配模式尽可能少的匹配所搜索的字符串 |
| . | 匹配除“\n”之外的任何单个字符 |
| (*pattern*) | 匹配*pattern*并获取这一匹配 |
| (?:*pattern*) | 匹配*pattern*但不获取匹配结果 |
| (?=*pattern*) | 正向预查，在任何匹配*pattern*的字符串开始处匹配查找字符串 |
| (?!*pattern*) | 负向预查，在任何不匹配*pattern*的字符串开始处匹配查找字符串 |
| *x*|*y* | 匹配*x*或*y*。例如，‘z|food’能匹配“z”或“food”。‘(z|f)ood’则匹配“zood”或“food” |
| [*xyz*] | 字符集合。匹配所包含的任意一个字符。例如，‘[abc]’可以匹配“plain”中的‘a’ |
| [^*xyz*] | 负值字符集合。匹配未包含的任意字符。例如，‘[^abc]’可以匹配“plain”中的‘p’ |
| [*a-z*] | 匹配指定范围内的任意字符。例如，‘[a-z]’可以匹配'a'到'z'范围内的任意小写字母字符 |
| [^*a-z*] | 匹配不在指定范围内的任意字符。例如，‘[^a-z]’可以匹配不在‘a’～‘z’'内的任意字符 |
| \b | 匹配一个单词边界，指单词和空格间的位置 |
| \B | 匹配非单词边界 |
| \d | 匹配一个数字字符，等价于[0-9] |
| \D | 匹配一个非数字字符，等价于[^0-9] |
| \f | 匹配一个换页符 |
| \n | 匹配一个换行符 |
| \r | 匹配一个回车符 |
| \s | 匹配任何空白字符，包括空格、制表符、换页符等 |

|  |  |
| --- | --- |
| \S | 匹配任何非空白字符 |
| \t | 匹配一个制表符 |
| \v | 匹配一个垂直制表符。等价于\x0b和\cK |
| \w | 匹配包括下划线的任何单词字符。等价于‘'[A-Za-z0-9\_]’ |
| \W | 匹配任何非单词字符。等价于‘[^A-Za-z0-9\_]’ |

由于在正则表达式中“\”、”？“、”\*“等字符已经具有了一定特殊意义，如果使用它们的原始意义，则应该对它进行转义。

1. 在C#中，要使用正则表达式类，需要添加命名空间：

Using system.Text.RegularExpressions;

1. RegX类常用的方法
2. 静态Match方法

使用静态Match方法，可以得到源中第一个匹配模式的连续子串

静态的Match方法有两个重载：

RegX.Match(string input,string pattern);

RegX.Match(string input,string pattern,RegexOptions options);

1. 静态的Matches方法

这个方法的重载形式同静态的Match方法，返回一个MatchCollection，表示输入中，匹配模式的匹配的集合。

1. 静态的IsMatch方法

此方法返回一个bool，重载形式同静态的Matches，如输入中匹配模式，返回TRUE，否则返回FALSE。

1. RegX类的实例

1.字符串替换

例如我想把如下格式记录中的NAME值修改为WANG

string line="ADDR=1234;NAME=ZHANG;PHONE=6789";

Regex reg = new Regex("NAME=(.+);");

string modified = reg.Replace(line, "NAME=WANG;");

修改后的字符串为 ADDR=1234;NAME=WANG;PHONE=6789

2.字符串匹配

例如我想提取刚才那条记录中的NAME值

Regex reg = new Regex("NAME=(.+);");

Match match=reg.Match(line);

string value=match.Groups[1].Value;

3、Match实例3

文本中含有"speed=30.2mph",需要提取该速度值，但是速度的单位可能是公制也可能是英制，mph,km/h,m/s都有可能；另外前后可能有空格。

string line="lane=1;speed=30.3mph;acceleration=2.5mph/s";

Regex reg=new Regex(@"speed\s\*=\s\*([\d\.]+)\s\*(mph|km/h|m/s)\*");

Match match=reg.Match(line);

那么在返回的结果中match.Groups[1].Value将含有数值，而match.Groups[2].Value将含有单位。

4、再比如，解码gps的GPRMC字符串，只需

Regex reg = new Regex(@"^\$GPRMC,[\d\.]\*,[A|V],(-?[0-9]\*\.?[0-9]+),([NS]\*),(-?[0-9]\*\.?[0-9]+),([EW]\*),.\*");

就可以获得经度、纬度值，而以前需要几十行代码。

1. System.Text.RegularExpressions命名空间的说明

该名称空间包括8个类，1个枚举，1个委托。他们分别是：

Capture: 包含一次匹配的结果；

CaptureCollection: Capture的序列；

Group: 一次组记录的结果，由Capture继承而来；

GroupCollection：表示捕获组的集合

Match: 一次表达式的匹配结果，由Group继承而来；

MatchCollection: Match的一个序列；

MatchEvaluator: 执行替换操作时使用的委托；

Regex：编译后的表达式的实例。

RegexCompilationInfo：提供编译器用于将正则表达式编译为独立程序集的信息 RegexOptions 提供用于设置正则表达式的枚举值

**Regex类中还包含一些静态的方法：**

Escape: 对字符串中的regex中的转义符进行转义；

IsMatch: 如果表达式在字符串中匹配，该方法返回一个布尔值；

Match: 返回Match的实例；

Matches: 返回一系列的Match的方法；

Replace: 用替换字符串替换匹配的表达式；

Split: 返回一系列由表达式决定的字符串；

Unescape:不对字符串中的转义字符转义。