字符串和正则表达式

siki 微信：devsiki QQ:804632564

字符串学习

1，创建字符串

System.String(string是这个类的别名)

System.Text.StringBuilder

2，正则表达式

System.String类

1,创建字符串　　string s = "www.devsiki.com";

2,获取字符串长度　s.Length(属性)

3,比较字符串是否一样 s=="www.devsiki.com"

4,字符串连接　　s="http://"+s;

5,使用类似索引器的语法来取得字符串中的某个字符 stringName[index]　s[0] s[3]

关于string字符串：string创建的字符串实际上是一个不可变的数据类型，一旦对字符串对象进行了初始化，该字符串就不能改变内容了，上面的示例中实际上是创建了一个新的字符串，把旧字符串的内容复制到新字符串中。然后把新字符串的引用赋值为字符串的对象。(重复修改给定的字符串，效率会很低)

关于字符串的更多方法

1,CompareTo()方法，比较字符串的内容

2,Replace()用另一个字符或者字符串替换字符串中给定的字符或者字符串

3,Split()在出现给定字符的地方，把字符串拆分称一个字符串数组

4,SubString()在字符串中检索给定位置的子字符串

5,ToLower()把字符串转换成小写形式

6,ToUpper()把字符串转换成大写形式

7,Trim()删除首尾的空白

8,Concat()方法，合并字符串

9,CopyTo()方法，把字符串中指定的字符复制到一个数组中

10,Format()方法，格式化字符串

11,IndexOf()方法，取得字符串第一次出现某个给定字符串或者字符的位置

12,IndexOfAny()方法，

13,Insert()把一个字符串实例插入到另一个字符串实例的制定索引处

14,Join()合并字符串数组，创建一个新字符串

StringBuilder类(位于System.Text命名空间下)

1,创建StringBuilder对象

```
StringBuilder sb = new StringBuilder("www.taikr.com");
StringBuilder sb = new StringBuilder(20);
StringBuilder sb = new StringBuilder("www.devsiki.com",100);
```

关于StringBuilder对象创建的时候的内存占用

2,Append()方法，给当前字符串追加一个字符

3,Insert()追加特定格式的字符串

4,Remove()从当前字符串中删除字符

5,Replace()在当前字符串中，用某个字符或者字符串全部替换另一个字符或者字符串

6,ToString()把当前stringBuilder中存储的字符串，提取成一个不可变的字符串

正则表达式

什么是正则表达式?

英文Regular Expression,是计算机科学的一个重要概念，她使用一种数学算法来解决计算机程序中的文本检索，匹配等问题，正则表达式语言是一种专门用于字符串处理的语言。在很多语言中都提供了对它的支持，c#也不例外，它可以帮我们解决下面的问题：

1，检索：通过正则表达式，从字符串中获取我们想要的部分

2，匹配：判断给定的字符串是否符合正则表达式的过滤逻辑

你可以认为正则表达式表述了一个字符串的书写规则

判断用户输入的密码是否合法，判断用户输入的邮箱格式是否合法

正则表达式的组成

正则表达式就是由普通字符以及特殊字符（成为元字符）组成的文字模式。该模式描述在查找文字主体时待匹配的一个或多个字符串。

元字符

见word文档

常用的操作正则表达式的方法和委托

下面学习一下位于System.Text.RegularExpressions下的Regex类的一些静态方法和委托

1,静态方法IsMatch （返回值是一个布尔类型,用于判断指定的字符串是否与正则表达式字符串匹配，它有三个重载方法）

```
bool IsMatch(string input, string pattern);
参数:       input:       要搜索匹配项的字符串。
            pattern:     要匹配的正则表达式模式。
            返回结果:  如果正则表达式找到匹配项，则为 true；否则，为 false。
bool IsMatch(string input, string pattern, RegexOptions options);
参数:       input:       要搜索匹配项的字符串。
            pattern:     要匹配的正则表达式模式。
            options:     枚举值的一个按位组合，这些枚举值提供匹配选项。
            返回结果:  如果正则表达式找到匹配项，则为 true；否则，为 false。
bool IsMatch(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
参数:       input:       要搜索匹配项的字符串。
            pattern:     要匹配的正则表达式模式。
            options:     枚举值的一个按位组合，这些枚举值提供匹配选项。
            matchTimeout: 超时间隔，或 System.Text.RegularExpressions.Regex.InfiniteMatchTimeout 指示该方法不应超时。
            返回结果:  如果正则表达式找到匹配项，则为 true；否则，为 false。
```

关于参数RegexOptions

它是一个枚举类型，有以下枚举值

RegexOptions枚举值	内联标志	简单说明
ExplicitCapture	n	只有定义了命名或编号的组才捕获
IgnoreCase	i	不区分大小写
IgnorePatternWhitespace	x	消除模式中的非转义空白并启用由 # 标记的注释。
MultiLine	m	多行模式，其原理是修改了^和$的含义
SingleLine	s	单行模式，和MultiLine相对应

内敛标志可以更小力度（一组为单位）的定义匹配选项

静态方法Match（System.Text.RegularExpressions）

静态方法Match，使用指定的匹配选项在输入字符串中搜索指定的正则表达式的第一个匹配项。 返回一个包含有关匹配的信息的对象。同样有三个重载方法，参数和IsMatch方法相同。此外，在Regex类中，还有一个同名的非静态方法，适用于多个实例的情况下，效率更高一些。

```
Match Match(string input, string pattern);
Match Match(string input, string pattern, RegexOptions options);
Match Match(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
```

静态方法Matches（System.Text.RegularExpressions）

静态方法Matches,在指定的输入字符串中搜索指定的正则表达式的所有匹配项。跟上面方法不同之处，就是这个方法返回的是所有匹配项，他同样有三个重载方法，并且参数和Match方法完全相同

```
MatchCollection Matches(string input, string pattern);
MatchCollection Matches(string input, string pattern, RegexOptions options);
MatchCollection Matches(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
```

Replaces函数（System.Text.RegularExpressions）

我们知道正则表达式主要是实现验证,提取,分割,替换字符的功能.Replace函数是实现替换功能的.

1)Replace(string input,string pattern,string replacement)

//input是源字符串,pattern是匹配的条件,replacement是替换的内容,就是把符合匹配条件pattern的内容转换成它

比如string result = Regex.Replace("abc", "ab", "##");

//结果是##c,就是把字符串abc中的ab替换成##

2)Replace(string input,string pattern,string replacement,RegexOptions options)

//RegexOptions是一个枚举类型,用来做一些设定.

//前面用注释时就用到了RegexOptions.IgnorePatternWhitespace.如果在匹配时忽略大小写就可以用RegexOptions.IgnoreCase

比如string result = Regex.Replace("ABc", "ab", "##",RegexOptions.IgnoreCase);

如果是简单的替换用上面两个函数就可以实现了.但如果有些复杂的替换,比如匹配到很多内容,不同的内容要替换成不同的字符.就需要用到下面两个函数

Replaces函数（System.Text.RegularExpressions）

3)Replace(string input,string pattern,MatchEvaluator evaluator);

//evaluator是一个代理,其实简单的说是一个函数指针,把一个函数做为参数参进来

//由于C#里没有指针就用代理来实现类似的功能.你可以用代理绑定的函数来指定你要实现的复杂替换.

4)Replace(string input,string pattern,MatchEvaluator evaluator,RegexOptions options);

//这个函数上上面的功能一样,只不过多了一点枚举类型来指定是否忽略大小写等设置

静态方法Split拆分文本

使用正则表达式匹配的位置，将文本拆分为一个字符串数组,同样有三个重载方法，返回值为字符串数组

```
string[] Split(string input, string pattern);

string[] Split(string input, string pattern, RegexOptions options);

string[] Split(string input, string pattern, RegexOptions options, TimeSpan matchTimeout);
```

@符号

我们经常在正则表达式字符串前面加上@字符，这样不让编译器去解析其中的转义字符，而作为正则表达式的语法（元字符）存在。

```
string s =@"www.baidu.com \n lkjsdflkj";
```

定位元字符

我们经常在正则表达式字符串前面加上@字符，这样不让编译器去解析其中的转义字符，而作为正则表达式的语法（元字符）存在。

字符	说明
\b	匹配单词的开始或结束
\B	匹配非单词的开始或结束
^	匹配必须出现在字符串的开头或行的开头
$	匹配必须出现在以下位置：字符串结尾、字符串结尾处的 \n 之前或行的结尾。
\A	指定匹配必须出现在字符串的开头（忽略 Multiline 选项）。
\z	指定匹配必须出现在字符串的结尾（忽略 Multiline 选项）。
\z	指定匹配必须出现在字符串的结尾或字符串结尾处的 \n 之前（忽略 Multiline 选项）。
\G	指定匹配必须出现在上一个匹配结束的地方。与 Match.NextMatch() 一起使用时，此断言确保所有匹配都是连续的。

定位元字符示例

示例一： 区配开始 ^

```
string str = "I am Blue cat";
Console.WriteLine(Regex.Replace(str, "^","准备开始:"));
```

示例二： 区始结束 $

```
string str = "I am Blue cat";
Console.WriteLine(Regex.Replace(str, "$", "  结束了！"));
```

基本语法元字符

字符	说明
.	匹配除换行符以外的任意字符
\w	匹配字母、数字、下划线、汉字 （指大小写字母、0-9的数字、下划线_）
\W	\w的补集 （ 除“大小写字母、0-9的数字、下划线_”之外）
\s	匹配任意空白符 （包括换行符/n、回车符/r、制表符/t、垂直制表符/v、换页符/f）
\S	\s的补集（除\s定义的字符之外）
\d	匹配数字（0-9数字）
\D	表示\d的补集 （除0-9数字之外）

在正则表达式中，\是转义字符. * 是元字符 如果要表示一个\ . *字符的话，需要使用\\ \. *

示例

示例一：校验只允许输入数字

```
string strCheckNum1 = "23423423a3", strCheckNum2 = "324234";
Console.WriteLine("匹配字符串"+strCheckNum1+"是否为数字:"+Regex.IsMatch(strCheckNum1, @"^\d*$"));
Console.WriteLine("匹配字符串" + strCheckNum2 + "是否为数字:" + Regex.IsMatch(strCheckNum2, @"^\d*$"));
```

示例二：校验只允许输入除大小写字母、0-9的数字、下划线_以外的任何字

```
string strCheckStr1 = "abcds_a", strCheckStr2 = "**&&((((2", strCheckStr3 = "**&&((((";
string regexStr = @"^\W*$";
Console.WriteLine("匹配字符串" + strCheckStr1 + "是否为除大小写字母、0-9的数字、下划线_以外的任何字符:" + Regex.IsMatch(strCheckStr1, regexStr));
Console.WriteLine("匹配字符串" + strCheckStr2 + "是否为除大小写字母、0-9的数字、下划线_以外的任何字符:" + Regex.IsMatch(strCheckStr2, regexStr));
Console.WriteLine("匹配字符串" + strCheckStr3 + "是否为除大小写字母、0-9的数字、下划
```

反义字符

字符	说明
\W	\w的补集 （ 除“大小写字母、0-9的数字、下划线_”之外）
\S	\s的补集 (除\s定义的字符之外)
\D	表示\d的补集 (除0-9数字之外)
\B	匹配不是单词开头或结束的位置
[ab]	匹配中括号中的字符
[a-c]	a字符到c字符之间是字符
[^x]	匹配除了x以外的任意字符
[^adwz]	匹配除了adwz这几个字符以外的任意字符

```
//示例：查找除ahou这之外的所有字符
string strFind1 = "I am a Cat!", strFind2 = "My Name's Blue cat!";
Console.WriteLine("除ahou这之外的所有字符，原字符为：" + strFind1 + "替换后:" +
Regex.Replace(strFind1, @"[^ahou]","*"));
Console.WriteLine("除ahou这之外的所有字符，原字符为：" + strFind2 + "替换后:" +
Regex.Replace(strFind2, @"[^ahou]", "*"));
```

重复描述字符

字符	说明
{n}	匹配前面的字符n次
{n,}	匹配前面的字符n次或多于n次
{n,m}	匹配前面的字符n到m次
?	重复零次或一次
+	重复一次或更多次
*	重复零次或更多次

示例：校验输入内容是否为合法QQ号(备注：QQ号为5-12位数字)

```
string isQq1 = "1233", isQq2 = "a1233", isQq3 = "0123456789123", isQq4 = "556878544";
string regexQq = @"^\d{5,12}$";
Console.WriteLine(isQq1+"是否为合法QQ号(5-12位数字):" + Regex.IsMatch(isQq1,
regexQq));
Console.WriteLine(isQq2 + "是否为合法QQ号(5-12位数字):" + Regex.IsMatch(isQq2,
regexQq));
```

择一匹配

字符	说明
|	将两个匹配条件进行逻辑“或”（Or）运算。

示例一：查找数字或字母

```
string findStr1 = "ad(d3)-df";
string regexFindStr = @"[a-z]|\d";
string newStrFind=String.Empty;
MatchCollection newStr =  Regex.Matches(findStr1, regexFindStr);
 newStr.Cast<Match>().Select(m => m.Value).ToList<string>().ForEach(i => newStrFind
+= i);
Console.WriteLine(findStr1 + "中的字母和数字组成的新字符串为:" + newStrFind);
```

示例二：将人名输出（"zhangsan;lisi,wangwu.zhaoliu"）

```
string strSplit = "zhangsan;lisi,wangwu.zhaoliu";
string regexSplitstr = @"[;]|[,]|[.]";
Regex.Split(strSplit, regexSplitstr).ToList().ForEach(i => Console.WriteLine(i));
```

择一匹配

示例三：校验国内电话号码(支持三种写法校验 A. 010-87654321 B. (010)87654321 C.01087654321 D.010 87654321)

```
string TelNumber1 = "(010)87654321", TelNumber2 = "010-87654321", TelNumber3 = "01087654321",
        TelNumber4 = "09127654321", TelNumber5 = "010)87654321",TelNumber6="(010-87654321",
        TelNumber7="91287654321";

Regex RegexTelNumber3 = new Regex(@"\(0\d{2,3}\)[- ]?\d{7,8}|^0\d{2,3}[- ]?\d{7,8}$");
Console.WriteLine("电话号码 " + TelNumber1 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber1));
Console.WriteLine("电话号码 " + TelNumber2 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber2));
Console.WriteLine("电话号码 " + TelNumber3 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber3));
Console.WriteLine("电话号码 " + TelNumber4 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber4));
Console.WriteLine("电话号码 " + TelNumber5 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber5));
Console.WriteLine("电话号码 " + TelNumber6 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber6));
Console.WriteLine("电话号码 " + TelNumber7 + " 是否合法:" + RegexTelNumber3.IsMatch(TelNumber7));
```

对正则表达式分组()

用小括号来指定子表达式(也叫做分组)

示例一：重复单字符 和 重复分组字符

```
Console.WriteLine("请输入一个任意字符串，测试分组：");
string inputStr = Console.ReadLine();
string strGroup1 = @"a{2}";
Console.WriteLine("单字符重复2两次替换为22，结果为："+Regex.Replace(inputStr, strGroup1,"22"));
//重复 多个字符 使用（abcd）{n}进行分组限定
string strGroup2 = @"(ab\w{2}){2}";
Console.WriteLine("分组字符重复2两次替换为5555，结果为：" + Regex.Replace(inputStr, strGroup2, "5555"));
```

示例二：校验IP4地址（如：192.168.1.4，为四段，每段最多三位，每段最大数字为255，并且第一位不能为0）

```
string regexStrIp4 = @"^((2[0-4]\d|25[0-5]|[01]?\d\d?)\.){3}(2[0-4]\d|25[0-5]|[01]?\d\d?))$";
Console.WriteLine("请输入一个IP4地址：");
string inputStrIp4 = Console.ReadLine();
Console.WriteLine(inputStrIp4 + " 是否为合法的IP4地址：" + Regex.IsMatch(inputStrIp4, regexStrIp4));
Console.WriteLine("请输入一个IP4地址：");
string inputStrIp4Second = Console.ReadLine();
Console.WriteLine(inputStrIp4 + " 是否为合法的IP4地址：" + Regex.IsMatch(inputStrIp4Second,
regexStrIp4));
```

关注公众号

发布最新的视频，文章和教学资源

THANK

@siki 微信：devsiki QQ:804632564