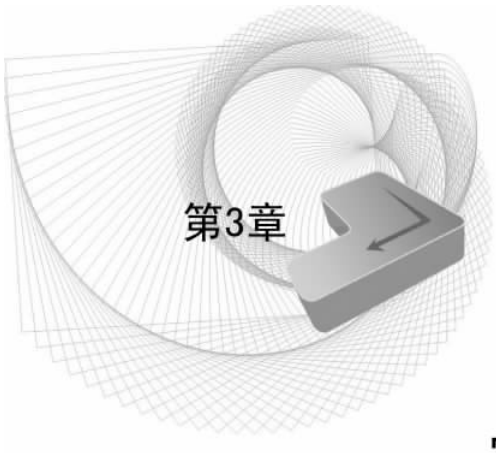

chapter 3

字符串与正则表达式

字符串操作是程序常用功能，用户自行实现字符串操作要设计大量的算法和复杂的数据结构，.NET 平台的 String 类具有字符串操作各项功能，Regex 类还支持字符串正则表达式的处理，这些功能为开发者操作字符串提供了便利。

3.1 .NET 平台中的 String 类

不同编程语言中实现字符串操作方式不同，.NET 平台提供的 String 类不仅具有面向对象性特点，而且处理能力强大。

3.1.1 字符与字符串

.NET 平台用关键字 char 来定义字符类型，这个 char 类型是用 16 位二进制表示的 Unicode 字符(与 Java 语言一致)，每个 char 类型变量使用两个字节。C 语言或者 C++语言的 char 类型是 8 位二进制数据，其变量使用一个字节，C 或者 C++语言中表示一个中文又使用两个字节，并且针对字符串会在所有字符内容的后面附一个值为\0 的字节，这些在字符串表示方面与 .NET 区别较大，因此 C 或者 C++中对字符串类型的经验算法不适用于 .NET 平台。在计算字符串长度的算法中 C 或 C++的结果与 C# 是完全不同的。

.NET 平台定义字符串为连续的 Unicode 字符，类型名为 System. String，string 是它的别名在程序中可不作区分的使用。.NET 程序员在操作字符串的需求例如字符查找，子串匹配，长度计算等只须直接调用 String 类的方法即可。字符串与简单数值类型如 Int32 型或 Double 类型对应值的转换也有相应的方法，例如 Int32. Parse()方法可将字符串转换为对应的整数值。

.NET 平台所有类都派生自 System. Object，当采用==符号比较两个实例的结果总是 false，==操作不比较对象的数值。String 类型不属简单类型而是派生自 System. Object 类，比较两个字符串对象的文本内容是否相等推荐使用 Compare 方法或者 CompareTo 方法，不过 .NET 平台极其特别的重载了字符串的==操作，当两个字符串值完全相同时“==”操作返回 true。

String 类的实例在创建后是只读的不可变的，即使程序不再需要，也无法进行垃圾回收，因此不宜存储密码等敏感信息，SecureString 对象能够自动加密，并能够删除其值。

3.1.2　字符串格式化输出

程序经常需要动态构造字符串即格式化输出，String 类的静态方法 Format 能够将复合格式项替换为文本形式的实例值。复合格式字符串由固定文本和索引占位符混合组成，其中索引占位符称为格式项，对应于列表中的对象。例如可以使用下面的格式字符串：

```
String.Format("hours = {0:hh}",DateTime.Now);
```

机器执行后 DateTime.Now 的数值就会转化为对应字符串代替原来{0:hh}的位置。{}中的数字值索引随后的参数列表，程序设计者不必编写将对象数值转化为对应文本的代码。如果要输出的结果中本身就存在{和}字符，代码中用{{或者}}表示。

3.1.3　@定义原义字符串

C＃语言与 C 和 C++语言等多数程序语言一样使用\作为转义字符，其后的字母代表不同的意义，C＃语言还引入一种新的方法处理字符串值包含转义字符的情况，它在源字符串前添加一个@字符，编译器会对引号中的字符串作原义处理，不执行转义操作，更方便写出字符串而无须考虑字符转义处理，例如一个完全限定的文件名：

```
@"c:\Docs\Source\a.txt"
```

这行可等价代替

```
"c:\\Docs\\Source\\a.txt"
```

若要在一个用@引出来的字符串内容包括双引号，需使用两对双引号：

```
@""""Ahoy!"" cried the captain."等价"Ahoy!" cried the captain.
```

3.1.4　静态方法与实例方法

String 类提供了成员函数来进行操作，表 3-1 列出 String 类的常用方法。

表 3-1　String 类的常用方法

方　法　名	用　　途	返 回 结 果
IndexOf	查找子串	匹配的子串下标，无则返回－1
Trim	移除首尾指定内容	移除指定内容后的字串
Split	切分字符串	指定元素分隔的字符串数组
Compare	比较字符串值	整数，表示比较结果，静态方法
CompareTo	比较字符串值	整数，代表比较结果，实例方法
Replace	替换指定子串	子串替换后的结果串
SubString	获取子串	按指定参数生成的子串

在面向对象程序语言中，类是复杂的数据结构，类的成员可以是数据也可以是方法，而对象是根据类的定义在程序运行时被创建出来的，由同一个类创建的多个对象占据不同的内存单元，对象也称为实例，这两种说法不加区分是等价的。类的成员可以使用 static 关键字标识，经过 static 关键字标识的成员则是静态成员，没有 static 关键字标识的成员是实例

成员，在程序运行时，实例成员是用对象名形式引用，静态方法则是以类名的引用形式。字符串类值比较有两个方法，分别是静态 Compare 方法和实例方法 CompareTo：

```
String a_str = "abcde";
String b_str = "12345";
```

String 类的静态方法是：

```
String.Compare(a_str,b_str)
```

String 对象实例方法形式是：

```
a_str.CompareTo(b_str);
b_str.CompareTo(a_str);
```

静态方法与实例方法的区别是：静态方法使用类名作为前导，实例方法则使用对象名作为前导。

3.1.5 使用 StringBuilder 类

String 对象的值在设定后是不可改变的，每次使用 System.String 类中的方法时，都要在内存中创建一个新的字符串对象，为新对象分配空间，在需要对字符串执行重复修改的情况下，创建新的 String 对象相关的系统开销可能会很大。如果要修改字符串而不创建新的对象，则可以使用 System.Text.StringBuilder 类提高程序执行效率。例如在一个循环中将许多字符串连接在一起时，使用 StringBuilder 类可以提升性能。通过重载的构造函数方法初始化变量，以下示例创建 StringBuilder 类的新实例。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!");
```

StringBuilder 是动态对象，能够对封装的字符串扩充字符的数量，可指定容纳的最大字符数。例如创建 StringBuilder 类其字符串为“Hello”(长度为 5)的实例，指定该对象的最大容量为 25。当修改 StringBuilder 时，在最大容量之前，不会重新分配空间。当达到容量时，自动分配新的空间且容量翻倍。下面重载的构造函数指定 MyStringBuilder 对象容量为 25。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!", 25);
```

使用 Capacity 属性来设置对象的最大长度，以下代码示例使用 Capacity 属性来定义对象的最大长度。

```
MyStringBuilder.Capacity = 25;
```

StringBuilder 类的常用方法如下。

(1) Append 方法可用来将文本或对象的字符串表示形式添加到由当前 StringBuilder 对象表示的字符串的结尾处。下面将一个 StringBuilder 对象初始化为“Hello World”，再向结尾追加文本。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!");
MyStringBuilder.Append(" What a beautiful day.");
```

示例的结果将会是：

"Hello World! What a beautiful day."。

(2) AppendFormat 方法将格式化文本添加到 StringBuilder 的结尾处，以下示例使用 AppendFormat 方法将一个格式化货币值追加到 StringBuilder 的结尾。

```
int MyInt = 25;
StringBuilder MyStringBuilder = new StringBuilder("Your total is ");
MyStringBuilder.AppendFormat("{0:C} ", MyInt);
```

运行结果是：

"Your total is ¥25.00"。

(3) Insert 方法将字符串或对象添加到当前 StringBuilder 中的指定位置，示例使用此方法将一个单词插入到 StringBuilder 的第 6 个位置。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!");
MyStringBuilder.Insert(6,"Beautiful ");
```

运行结果是：

"Hello Beautiful World!"。

(4) Remove 方法从指定的零开始的索引处 StringBuilder 中移除指定数量的字符，示例使用 Remove 方法移去部分字符。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!");
MyStringBuilder.Remove(5,7);
```

运行结果是：

"Hello"

(5) Replace 方法用另一个指定的字符来替换 StringBuilder 对象内的字符，示例查找原字符串中所有的感叹号字符“!”，并用问号字符“?”来替换它们。

```
StringBuilder MyStringBuilder = new StringBuilder("Hello World!");
MyStringBuilder.Replace('!', '?');
```

运行结果是：

"Hello World?"。

程序在使用非托管 API 时，参数传递及返回值特定情况下只能由 StringBuilder 类来完成任务，而 String 类无法胜任。

3.2 正则表达式

正则表达式用来描述或者匹配一系列符合某个句法规则的字符串，它由普通字符(例如字符 a 到 z)以及特殊字符(称为元字符)形成约定模式，该模式在目标字串中匹配一个或多

个字符组合。很多文本编辑器或其他软件工具都使用正则表达式来检索或替换那些符合某个模式的文本内容。

正则表达式的语法定义并不是编程语言，它遵从通用规则描述字符串模式，如 email 地址这样的字符串就具有一定模式，非常适合用正则表达式来识别和操作。许多编程语言支持正则表达式的字符串操作，.NET 平台具有正则表达式类，能够减少开发者自己编写字符串模式查找匹配的任务。

正则表达使用预定义的符号表示目标字符串的模式，例如“\w”表示一个单词的匹配，“\s”代表空白字符的匹配，“\d”代表十进制数字字符，“nD”代表非数字字符；在匹配的时候还可以限定匹配对象的数量，用“*”表示它前面的模式对象任意次出现，包括零次；采用“*”表示它前面的模式对象出现不少于 1 次；“n,m”表示它前面的模式对象出现次数可介于数值 n 和 m 之间。正则表达式中存在转义字符，应用“@”字符编写模式串会变得非常简便。正则表达式的复杂内容本书不作介绍，用户可参阅 MSDN。

3.2.1 正则表达式类

.NET 平台提供下面几个正则表达式操作类。

(1) Regex 类使用字符串表达的正则表达式语法创建一个正则表达式，它还有操作其他对象的静态方法。

(2) Match 类表示正则表达式匹配操作的结果，Match.Success 属性来指示是否已找到匹配。

(3) MatchCollection 类表示成功的非重叠匹配项的序列。

(4) GroupCollection 类表示在单个匹配项中返回该捕获组的集合。

(5) CaptureCollection 类表示捕获的子字符串的序列，并返回由单个捕获组所执行的捕获集。

(6) Group 类表示单个捕获组的结果。

3.2.2 使用正则表达式搜索字符串

.NET 平台正则表达式能简化字符串模式匹配编程，下面的程序片段建立一个正则表达式类，它查找给定的模式字符串“abc”，并给出查找到的位置，.NET 平台的 Regex 类查找特定模式字符串比程序员编写全部算法代码简单多了。这段代码使用到的类 MessageBox 可以便捷显示变量内容，它的静态方法 Show 会显示一个字符串，结合 String.Format 方法来显示程序运行中变量的值。

```
// Create a new Regex object
Regex r = new Regex("abc");
// Find a single match in the string
Match m = r.Match("123abc456");
if (m.Success)
{
  MessageBox.Show(String.Format("Found match at position {0}", m.Index));
}
```

以下程序示例使用正则表达式(Abc)+来查找字符串“XYZAbcAbcAbcXYZAbcAb”

中的一个或多个匹配。(Abc)+的意思是字符串“Abc”可出现最少一次的情况,该示例阐释了使用 Captures 属性来返回多组捕获的子字符串。

```
using System;
using System.Text.RegularExpressions;
private void button1_Click(object sender, EventArgs e)
{
int counter;
Match m;
CaptureCollection cc;
GroupCollection gc;
// Look for groupings of "Abc"
Regex r = new Regex("(Abc)+");
// Dene the string to search
m = r.Match("XYZAbcAbcAbcXYZAbcAb");
gc = m.Groups;
// Print the number of groups
MessageBox("Found match at position {0}", m.Index);
MessageBox.Show("Captured groups = " + gc.Count.ToString());
// Loop through each group
for (int i=0; i<gc.Count; i++)
{
  cc = gc[i].Captures;
  counter = cc.Count;
  // Print number of captures in this group
  MessageBox.Show("Captures count = " + counter.ToString());
  // Loop through each capture in group
  for (int ii = 0; ii<counter; ii++)
    {
      // Print capture and position
      MessageBox.Show(cc[ii] + " Starts at character " +cc[ii].Index);
    }
  }
}
```

运行结果如下:

```
Captured groups = 2
Captures count = 1
AbcAbcAbc Starts at character 3
Captures count = 3
Abc Starts at character 3
Abc Starts at character 6
Abc Starts at character 9
```

3.3 代码片段管理

程序设计者经常搜索大量的程序文件将已有的代码复制到当前的工作任务中,经过微小改动生成目标代码,重复性的代码输入工作不仅低效率也往往让人觉得乏味。VS9 与 VS12 都具有代码段管理的功能称为 Code Snippet,它先将一段代码命名后保存起来,然后

通过菜单命令快捷插入新程序中，这种方法提高代码段复用及代码编写效率。

工具 VS9 或 VS12 内置典型的代码片段，例如 for 语句片段、switch 语句片段等。例如输入 region 大纲命名这个代码段，在代码视图中使用右键弹出上下文菜单，选择插入代码段菜单项如图 3-1 所示，选择 Visual C# 目录如图 3-2 所示，继续选择 # region 选项如图 3-3 所示，代码段自动加入文件，如图 3-4 所示。

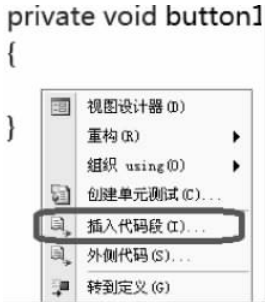

图 3-1 右键菜单项：插入代码段

图 3-2 插入代码段

图 3-3 插入代码段 # region

图 3-4 插入的 region 代码段

用户还可以将自己编写的或者他人编写的代码段导入 VS 工具中，代码片段是一种称为 Snippet 的 XML 格式的文件，有意义或者相关性的文件名指示代码段的功能，下面是一个 Snippet 文件的内容。

```
    XML Code
<?xml version = "1.0" encoding = "utf - 8" ?>
 < CodeSnippets xmlns = "http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  < CodeSnippet Format = "1.0.0">
   < Header >
     < Title > MessageBox.show </Title >
     < Shortcut > MessageBox </Shortcut >
     < Author > 李赞 </Author >
   </Header >
   < Snippet >
  < Code Language = "CSharp">
    <![CDATA[
     MessageBox.Show("Text");
    ]]>
  </Code >
  </Snippet >
```

```
 </CodeSnippet>
</CodeSnippets>
```

<Header>结点中最重要的是 Title 和 Shortcut 结点值，对应 Snippet 的名称和快捷键；它还可以包含若干个 SnippetType 结点，有 3 种取值：Expansion、SurroundsWith 和 Refactoring。Expansion 允许代码插入在光标处；SurroundsWith 允许代码围绕在选中代码两边(就像 # region 那样)；Refactoring 指定了在 C# 重构过程中所使用的 Snippet，在自定义 Snippet 中不能使用，如果该值不做设置，Snippet 代码段可以放在任何地方。

本书不对 Snippet 的 XML 文件格式作深入说明，读者只需修改 Title 部分 Shortcut 部分和 Code 部分，保持其他部分不变即可生成新的代码段。Title 内容用来在显示菜单时作为菜单项文本，Shortcut 内容表示菜单项的说明文本，而 Code 部分被插入到文件中的代码。应用 Code Snippet 能提高编程输入效率，例如使用正则表达来查找字符串：

```
// Create a new Regex object
Regex r = new Regex("abc");
// Find a single match in the string
Match m = r.Match("123abc456");
if (m.Success)
{
  MessageBox.show(String.Format("Found match at position {0}", m.Index));
}
```

修改原 Snippet 文件的 Title 部分、Shortcut 部分和 Code 部分为下面的内容：

```
XML Code
<?xml version = "1.0" encoding = "utf - 8" ?>
<CodeSnippets xmlns = "http://schemas.microsoft.com/VisualStudio/2005/CodeSnippet">
  <CodeSnippet Format = "1.0.0">
    <Header>
      <Title> 使用正则表达式查找文件 </Title>
      <Shortcut> 正则表达式 </Shortcut>
      <Author> 李赞 </Author>
    </Header>
  <Snippet>
    <Code Language = "CSharp">
    <![CDATA[
      // Create a new Regex object
      Regex r = new Regex("abc");
      // Find a single match in the string
      Match m = r.Match("123abc456");
      if (m.Success)
      { MessageBox.show(String.Format("Found match at position {0}", m.Index));
      }
      ]]>
      </Code>
    </Snippet>
  </CodeSnippet>
</CodeSnippets>
```

将文件保存为 regExp.snippet，选择“工具”→“代码段管理器”（或连按 Ctrl＋K、Ctrl＋B 键），打开 Code Snippets Manager 窗口，如图 3-5 所示，通过 Import 按钮来导入自定义的 Snippet，如图 3-6 所示，为代码段选择一个位置，这将决定其在代码编辑时上下文菜单中的选项位置如图 3-7 所示，工具包含正则表达式这个菜单。在代码文件中通过右键菜单调出上下文菜单，找到设定位置的新加的菜单项，如图 3-8 所示，一段事先设定的代码就会自动插入到当前位置，如图 3-9 所示。

图 3-5　代码段管理器

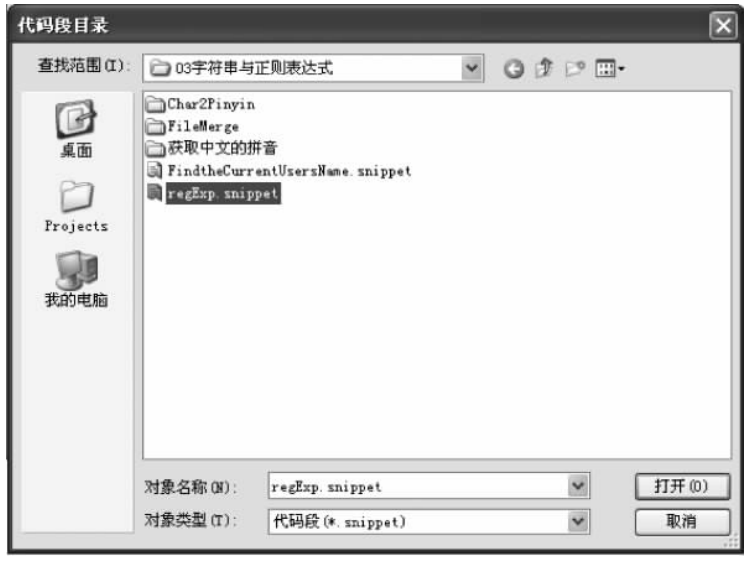

图 3-6　选择代码段文件

代码段管理小工具提高了代码编写效率，同时还带给程序员一些乐趣。

图 3-7　设置代码段管理位置

图 3-8　插入自定义代码段

```
// Create a new Regex object
Regex r = new Regex("abc");
// Find a single match in the string
Match m = r.Match("123abc456");
if (m.Success)
{
   MessageBox.show(String.Format("Found match at position {0}", m.Index));
}
```

图 3-9　自定义代码段效果

3.4　思考与练习

1. 使用正则表达式在目标字符中查找指定模式。
2. 创建一个自定义代码段，并在程序中使用。