C# 命名空间（Namespace） C# 正则表达式

C# 预处理器指令

预处理器指令指导编译器在实际编译开始之前对信息进行预处理。

所有的预处理器指令都是以 # 开始。且在一行上，只有空白字符可以出现在预处理器指令之前。预处理器指令不是语句，所以它们不以分号（;）结束。

C# 编译器没有一个单独的预处理器，但是，指令被处理时就像是有一个单独的预处理器一样。在 C# 中，预处理器指令用于在条件编译中起作用。与 C 和 C++ 不同的是，它们不是用来创建宏。一个预处理器指令必须是该行上的唯一指令。

C# 预处理器指令列表

下表列出了 C# 中可用的预处理器指令：

预处理器指令	描述
#define	它用于定义一系列成为符号的字符。
#undef	它用于取消定义符号。
#if	它用于测试符号是否为真。
#else	它用于创建复合条件指令，与 #if 一起使用。
#elif	它用于创建复合条件指令。
#endif	指定一个条件指令的结束。
#line	它可以让您修改编译器的行数以及（可选地）输出错误和警告的文件名。
#error	它允许从代码的指定位置生成一个错误。
#warning	它允许从代码的指定位置生成一级警告。
#region	它可以让您在使用 Visual Studio Code Editor 的大纲特性时，指定一个可展开或折叠的代码块。
#endregion	它标识着 #region 块的结束。

#define 预处理器

#define 预处理器指令创建符号常量。

#define 允许您定义一个符号，这样，通过使用符号作为传递给 #if 指令的表达式，表达式将返回 true。它的语法如下：

```
#define symbol
```

下面的程序说明了这点：

实例

```
#define PI
using System;
namespace PreprocessorDAppl
{
   class Program
   {
      static void Main(string[] args)
      {
         #if (PI)
            Console.WriteLine("PI is defined");
         #else
            Console.WriteLine("PI is not defined");
         #endif
         Console.ReadKey();
      }
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
PI is defined
```

条件指令

您可以使用 #if 指令来创建一个条件指令。条件指令用于测试符号是否为真。如果为真，编译器会执行 #if 和下一个指令之间的代码。

条件指令的语法：

```
#if symbol [operator symbol]...
```

其中，*symbol* 是要测试的符号名称。您也可以使用 true 和 false，或在符号前放置否定运算符。

常见运算符有：

- == (等于)
- != (不等于)
- && (与)
- || (或)

您也可以用括号把符号和运算符进行分组。条件指令用于在调试版本或编译指定配置时编译代码。一个以 **#if** 指令开始的条件指令，必须显示地以一个 **#endif** 指令终止。

下面的程序演示了条件指令的用法：

实例

```
#define DEBUG
#define VC_V10
using System;
public class TestClass
{
   public static void Main()
   {

      #if (DEBUG && !VC_V10)
         Console.WriteLine("DEBUG is defined");
      #elif (!DEBUG && VC_V10)
         Console.WriteLine("VC_V10 is defined");
      #elif (DEBUG && VC_V10)
         Console.WriteLine("DEBUG and VC_V10 are defined");
      #else
         Console.WriteLine("DEBUG and VC_V10 are not defined");
      #endif
      Console.ReadKey();
   }
}
```

当上面的代码被编译和执行时，它会产生下列结果：

```
DEBUG and VC_V10 are defined
```

C# 命名空间（Namespace） C# 正则表达式

1 篇笔记 写笔记

预处理器指令的用途理解：

在程序调试和运行上有重要的作用。比如预处理器指令可以禁止编译器编译代码的某一部分，如果计划发布两个版本的代码，即基本版本和有更多功能的企业版本，就可以使用这些预处理器指令来控制。在编译软件的基本版本时，使用预处理器指令还可以禁止编译器编译于额外功能相关的代码。另外，在编写提供调试信息的代码时，也可以使用预处理器指令进行控制。总的来说和普通的控制语句（if等）功能类似，方便在于预处理器指令包含的未执行部分是不需要编译的。

```
#define PI
using System;
namespace PreprocessorDAppl
{
   class Program
   {
      static void Main(string[] args)
      {
         #if (PI)
            Console.WriteLine("PI is defined");     //PI不存在，则这条语句不编译
```

```
        #else
           Console.WriteLine("PI is not defined"); //PI存在，则这条语句不编译
        #endif
        Console.ReadKey();
      }
   }
}
```

其他预处理器指令：

#warning 和 #error：

当编译器遇到它们时，会分别产生警告或错误。如果编译器遇到 #warning 指令，会给用户显示 #warning 指令后面的文本，之后编译继续进行。如果编译器遇到 #error 指令，就会给用户显示后面的文本，作为一条编译错误消息，然后会立即退出编译。使用这两条指令可以检查 #define 语句是不是做错了什么事，使用 #warning 语句可以提醒自己执行某个操作。

```
#if DEBUG && RELEASE
#error "You've defined DEBUG and RELEASE simultaneously!"
#endif
#warning "Don't forget to remove this line before the boss tests the code!"
Console.WriteLine("*I hate this job.*");
```

2. #region 和 #endregion

#region 和 #endregion 指令用于把一段代码标记为有给定名称的一个块，如下所示:

```
#region Member Field Declarations
int x;
double d;
Currency balance;
#endregion
```

这看起来似乎没有什么用，它不影响编译过程。这些指令的优点是它们可以被某些编辑器识别，包括 Visual Studio .NET 编辑器。这些编辑器可以使用这些指令使代码在屏幕上更好地布局。

3. #line

#line 指令可以用于改变编译器在警告和错误信息中显示的文件名和行号信息，不常用。

如果编写代码时，在把代码发送给编译器前，要使用某些软件包改变输入的代码，就可以使用这个指令，因为这意味着编译器报告的行号或文件名与文件中的行号或编辑的文件名不匹配。#line指令可以用于还原这种匹配。也可以使用语法#line default把行号还原为默认的行号：

```
#line 164 "Core.cs" // 在文件的第 164 行
// Core.cs, before the intermediate
// package mangles it.
// later on
#line default // 恢复默认行号
```

4. #pragma

#pragma 指令可以抑制或还原指定的编译警告。与命令行选项不同，#pragma 指令可以在类或方法级别执行，对抑制警告的内容和抑制的时间进行更精细的控制。如下：

```
#pragma warning disable 169    // 取消编号 169 的警告（字段未使用的警告）
public class MyClass
{
    int neverUsedField;       // 编译整个 MyClass 类时不会发出警告
}
#pragma warning restore 169   // 恢复编号 169 的警告
```

樱花树 9个月前 (06-08)