Visual Studio 教程 | C#

使用 Visual Studio 创建 C# 应用。

入门

操作指南

下载并安装 Visual Studio

开始使用

启动引导式教程

快速入门

创建“Hello World”Web 应用

创建 C# 应用

教程

创建控制台应用

创建 Web 应用

创建 Windows App SDK 应用

创建 Windows 桌面应用

创建 Windows 窗体应用程序

了解 C# 语言

Windows 窗体匹配游戏

Windows 窗体数学测验

Windows 窗体图片查看器

学习 Visual Studio

教程

运行程序

从存储库打开代码

编写和编辑代码

编译和生成

调试代码

测试代码

操作指南

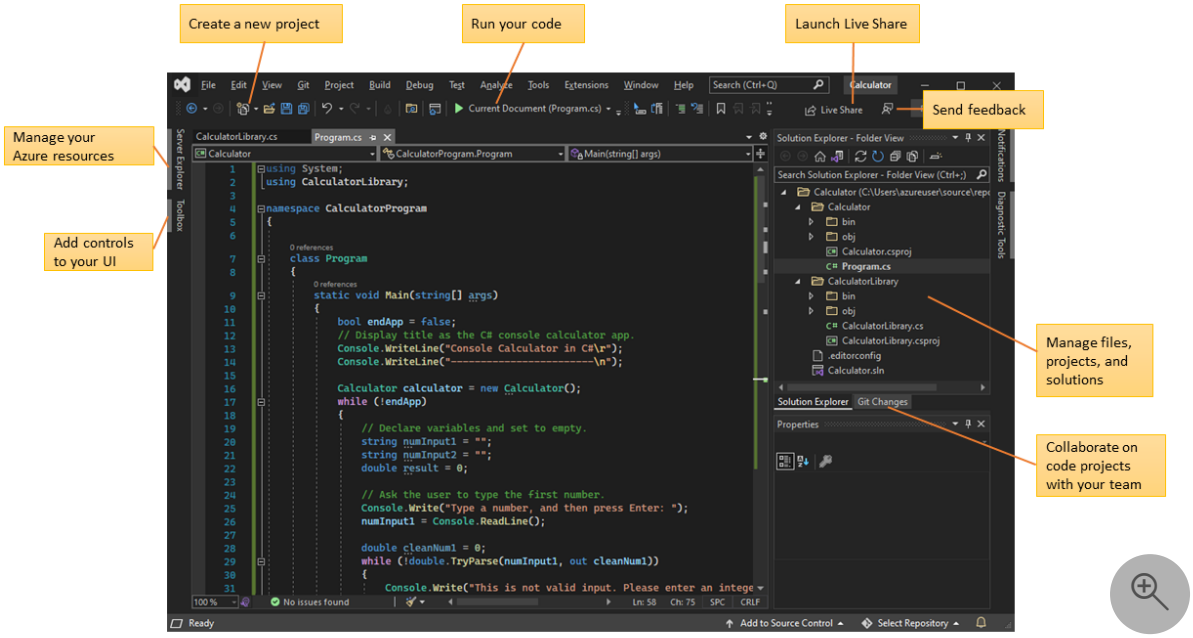
访问数据

欢迎使用 Visual Studio IDE | C#

项目 • 2023/03/10

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

集成开发环境 (IDE) 是一个功能丰富的程序，支持软件开发的许多方面。 Visual Studio IDE 是一种创新启动板，可用于编辑、调试并生成代码，然后发布应用。 除了大多数 IDE 提供的标准编辑器和调试器之外，Visual Studio 还包括编译器、代码完成工具、图形设计器和许多其他功能，以加强软件开发过程。



上图显示了 Visual Studio 和一个打开的项目，其中显示了关键窗口及其功能：

- 在“解决方案资源管理器”右上角，可以查看、导航和管理代码文件。 解决方案资源管理器可将代码文件分组为解决方案和项目，从而帮助整理代码 。
- 中心编辑器窗口用于显示文件内容，你的大部分时间可能都是在此窗口中度过的。 在编辑窗口中，可以编辑代码或设计用户界面，例如带有按钮和文本框的窗口。
- 在 Git 更改右下方，可使用版本控制技术（如 Git 和 GitHub ）跟踪工作项并与他人共享代码。

版本

Visual Studio 适用于 Windows 和 Mac。 Visual Studio for Mac 的许多功能与 Visual Studio for Windows 相同，并针对开发跨平台应用和移动应用进行了优化。 本文重点介绍 Visual Studio 的 Windows 版本。

Visual Studio 有三个版本：社区版、专业版和企业版。 请参阅比较 Visual Studio 版本，了解各个版本支持的功能。

高效性方面的常用功能

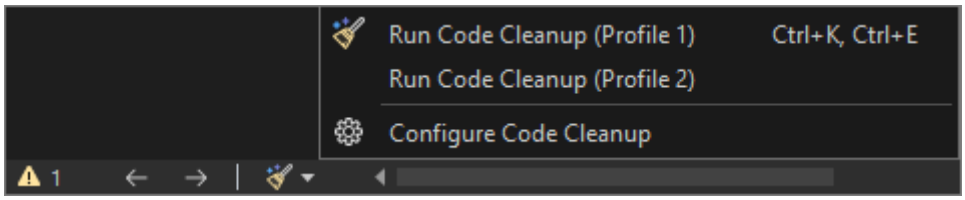
开发软件时，Visual Studio 可帮助提高工作效率的一些常用功能包括：

- 波形曲线和快速操作

  波形曲线是波浪形下划线，它可以在键入时对代码中的错误或潜在问题发出警报。 这些视觉线索可帮助你立即解决问题，而无需等待在生成或运行时发现错误。 如果将鼠标悬停在波形曲线上，将看到关于此错误的更多信息。 灯泡也可能显示在左边距中，其中显示可采取以修复错误的快速操作。

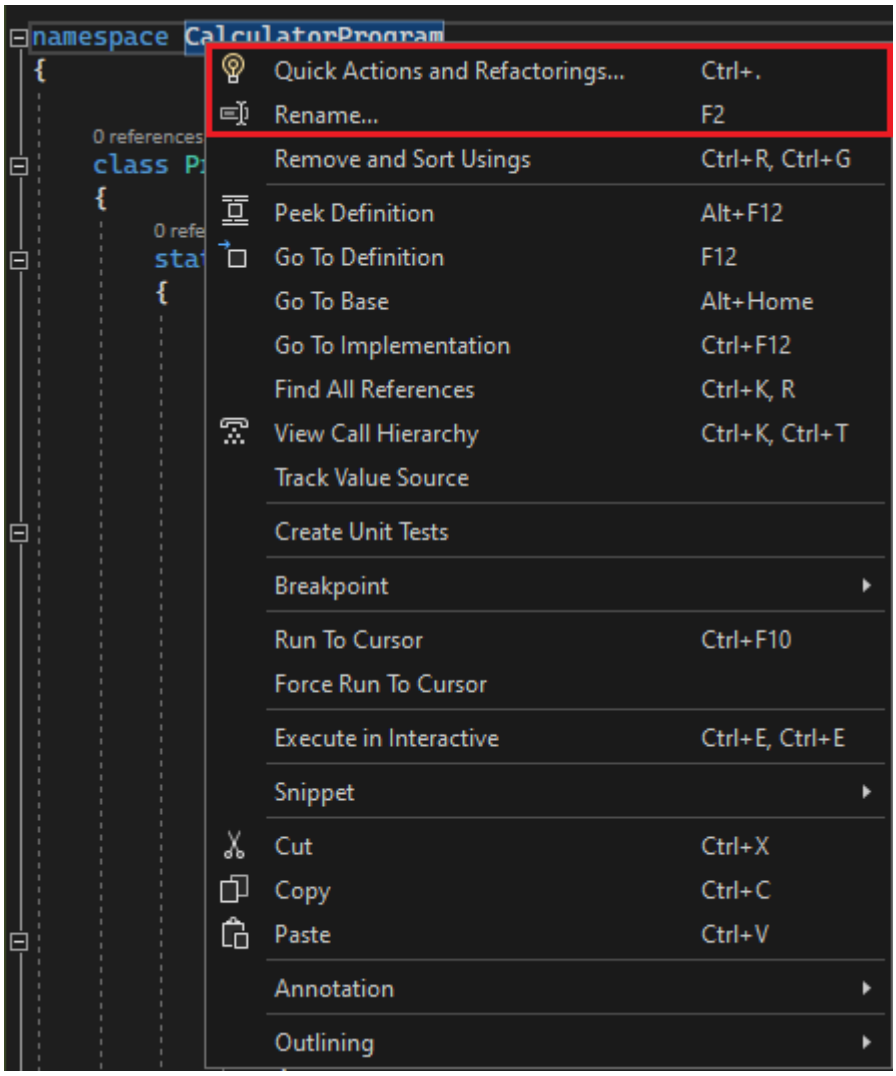
- 代码清理

  通过单击一个按钮，可以设置代码格式并应用代码样式设置、.editorconfig 约定和 Roslyn 分析器建议的任何代码修复程序。 代码清理（当前只适用于 C# 代码）有助于在代码进入代码评审之前解决代码中的问题。



- 重构

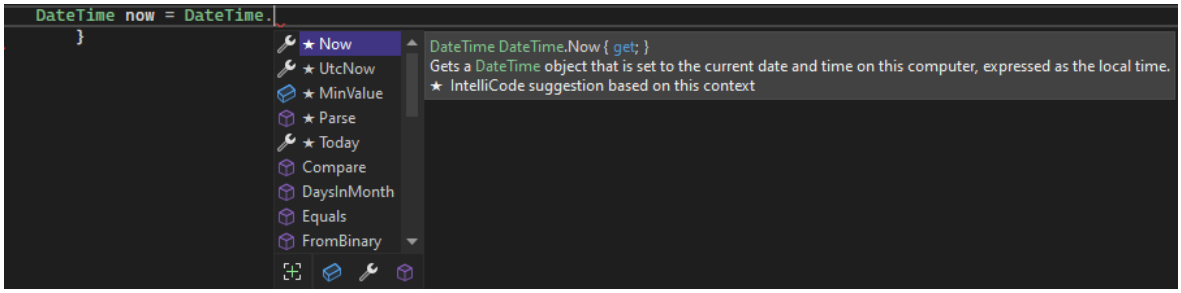
  重构包括智能重命名变量、将一个或多个代码行提取到新方法中和更改方法参数的顺序。



- IntelliSense

  IntelliSense 是一组功能，可用于在编辑器中直接显示代码的信息，并且可在某些情况下编写小段代码。 如同在编辑器中拥有了基本文档内联，从而无需在其他位置查看类型信息。

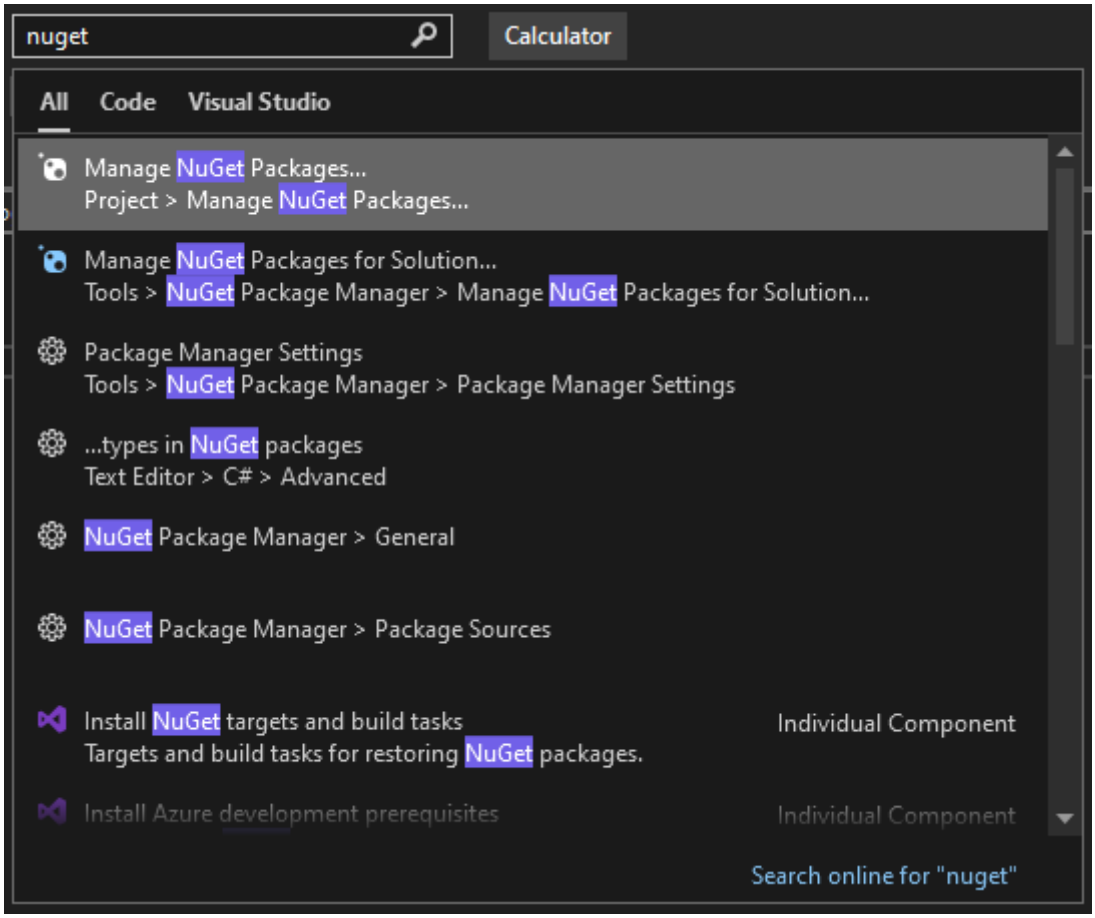
  下图显示了 IntelliSense 如何显示类型的成员列表：



  IntelliSense 功能因语言而异。 有关详细信息，请参阅 C# IntelliSense、Visual C++ IntelliSense、JavaScript IntelliSense 和 Visual Basic IntelliSense。

- Visual Studio 搜索

Visual Studio 菜单、选项和属性有时可能会让人不知所措。 Visual Studio 搜索或 Ctrl+Q 是在同一位置快速查找 IDE 功能和代码的绝佳方法。



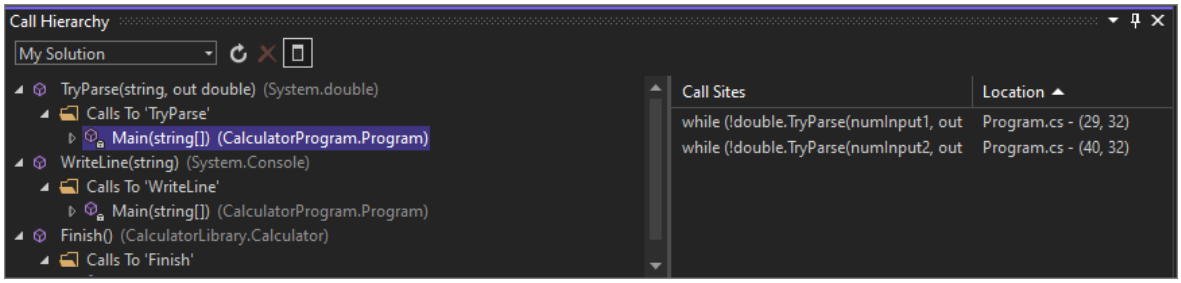
有关信息和工作效率提示，请参阅如何使用 Visual Studio 搜索。

- Live Share

  与他人实时协作编辑和调试，无需考虑应用类型或编程语言。 可以立即安全地共享项目。 还可以共享调试会话、终端实例、localhost Web 应用、语音呼叫等等。

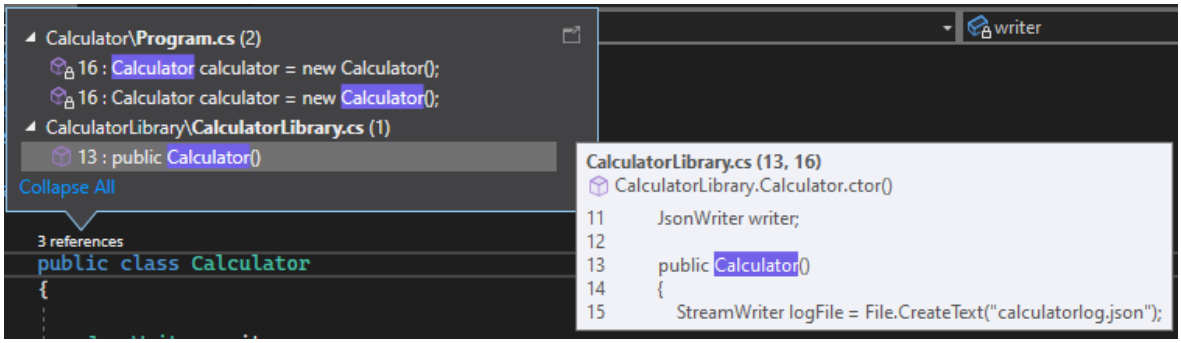
- 调用层次结构

  “调用层次结构”窗口显示调用所选方法的方法。 考虑更改或删除方法时，或者尝试追踪 bug 时，这可能是有用的信息。



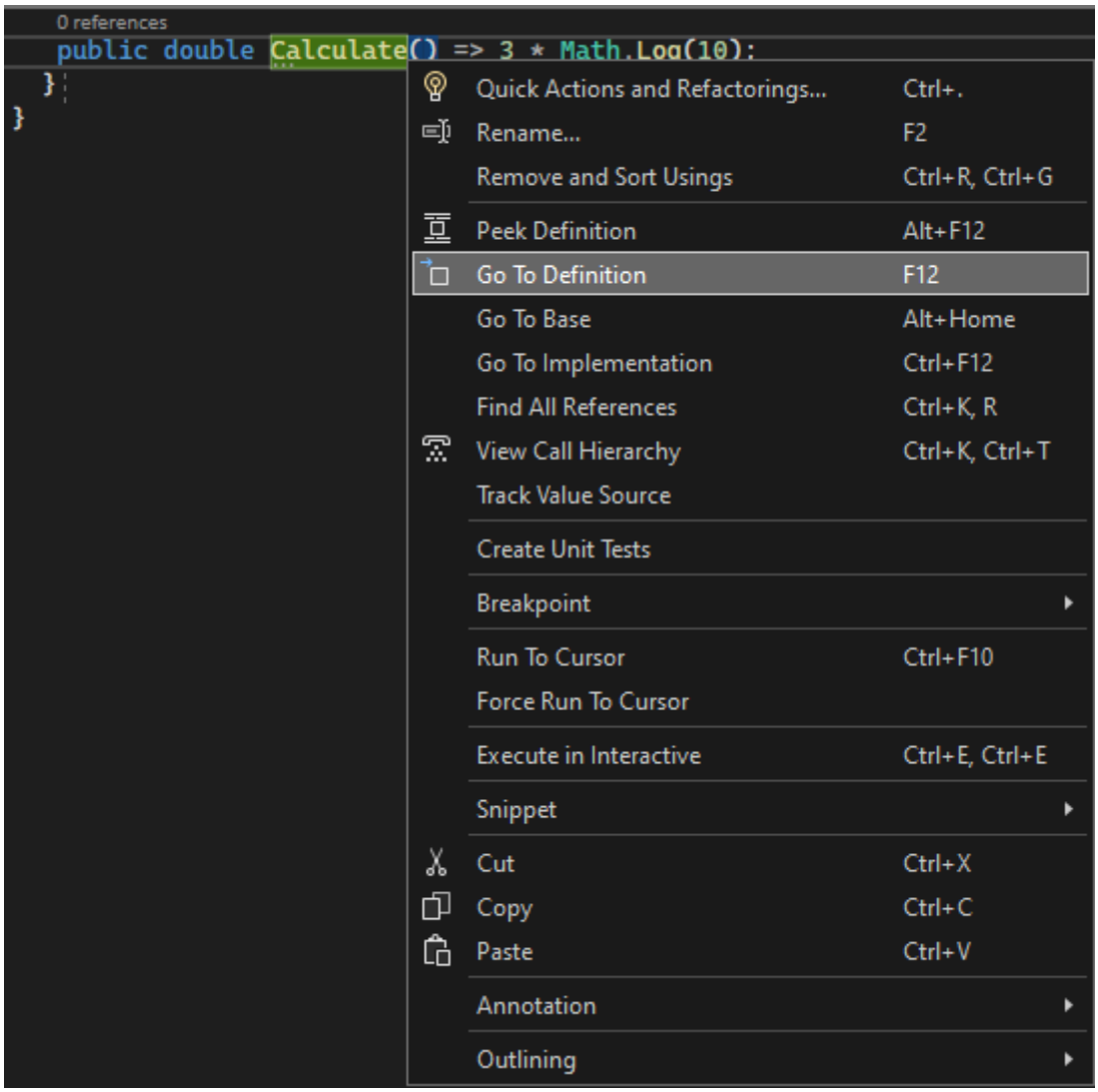
- CodeLens

CodeLens 可帮助查找代码引用、代码更改、链接错误、工作项、代码评审和单元测试，所有操作都在编辑器上进行。



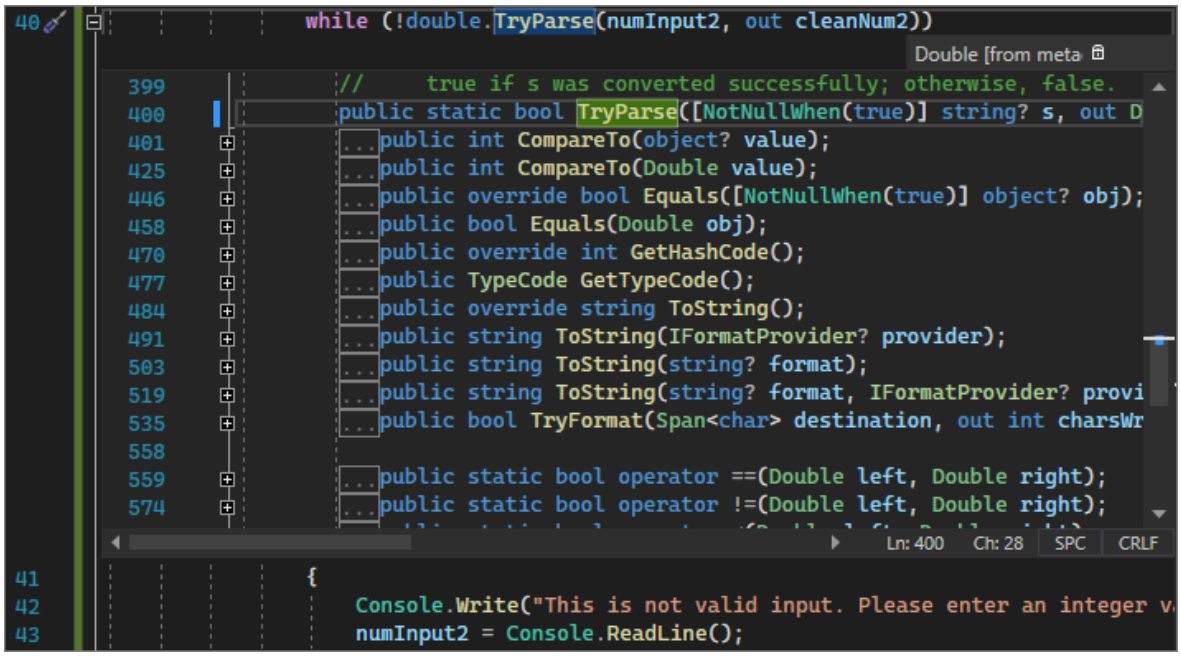
- 转到定义

“转到定义”功能可将你直接带到函数或类型定义的位置。



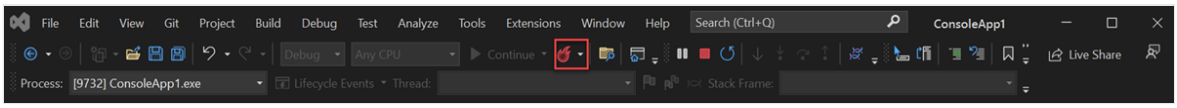
- 查看定义

“速览定义”窗口显示方法或类型定义，而无需打开一个单独的文件。



- 热重载

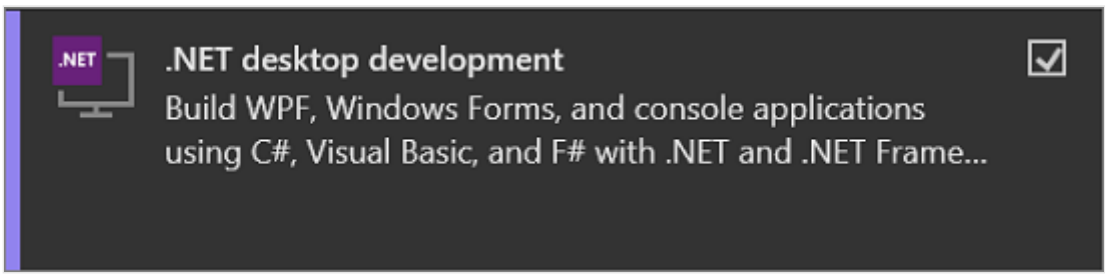
  使用热重载，可以编辑应用程序的代码文件，并将代码更改立即应用于正在运行的应用程序。



安装 Visual Studio

在本部分中，你将创建一个简单的项目来尝试可在 Visual Studio 中执行的一些操作。 你将使用 IntelliSense 作为编码辅助，调试应用以便在应用执行期间查看变量值，并更改颜色主题。

首先，请下载 Visual Studio 并将其安装到你的系统上。 在模块化安装程序中，可以选择和安装工作负载。工作负载是你希望使用的编程语言或平台所需的一些功能。 若要使用以下步骤创建程序，请确保在安装过程中选择“.NET 桌面开发”工作负载。



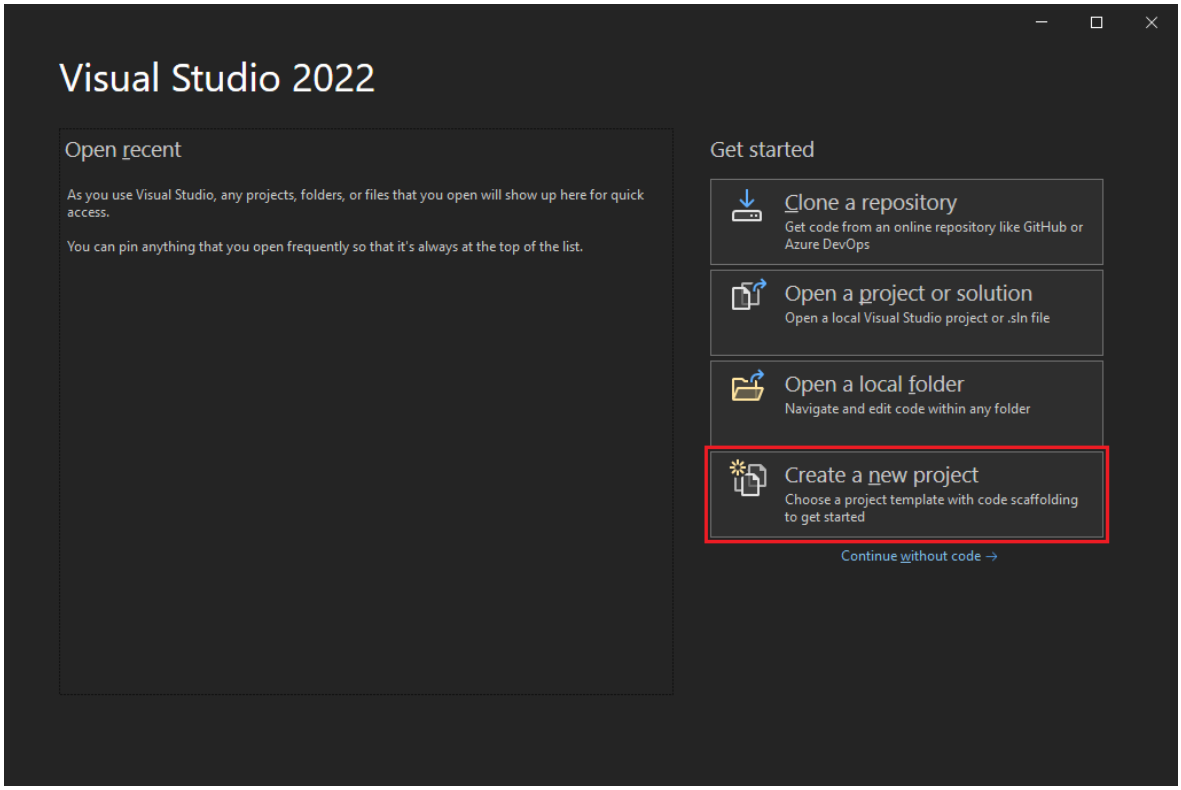
首次打开 Visual Studio 时，可使用 Microsoft 帐户或者单位或学校帐户登录。

创建程序

深入了解并创建一个简单的程序。

1. 启动 Visual Studio。 “启动”窗口中会显示有关克隆存储库、打开最近的项目或创建新项目的选项。

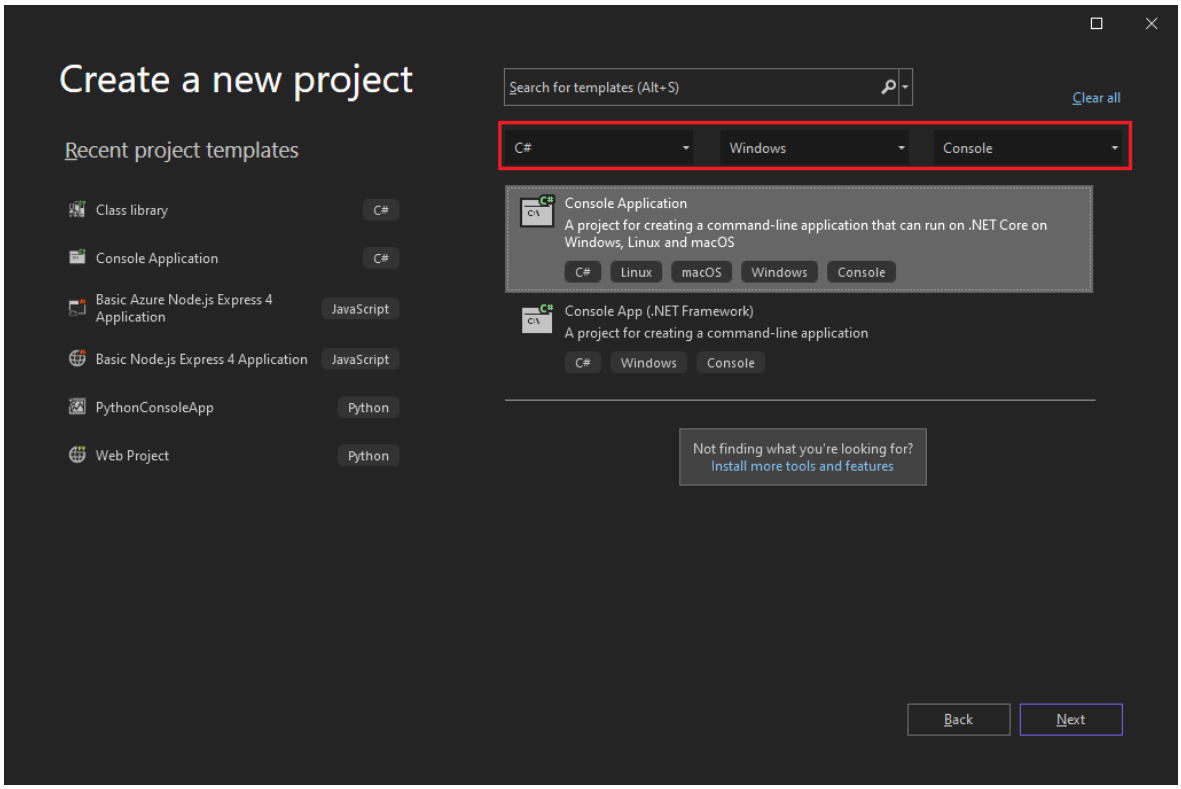
2. 选择“创建新项目”。



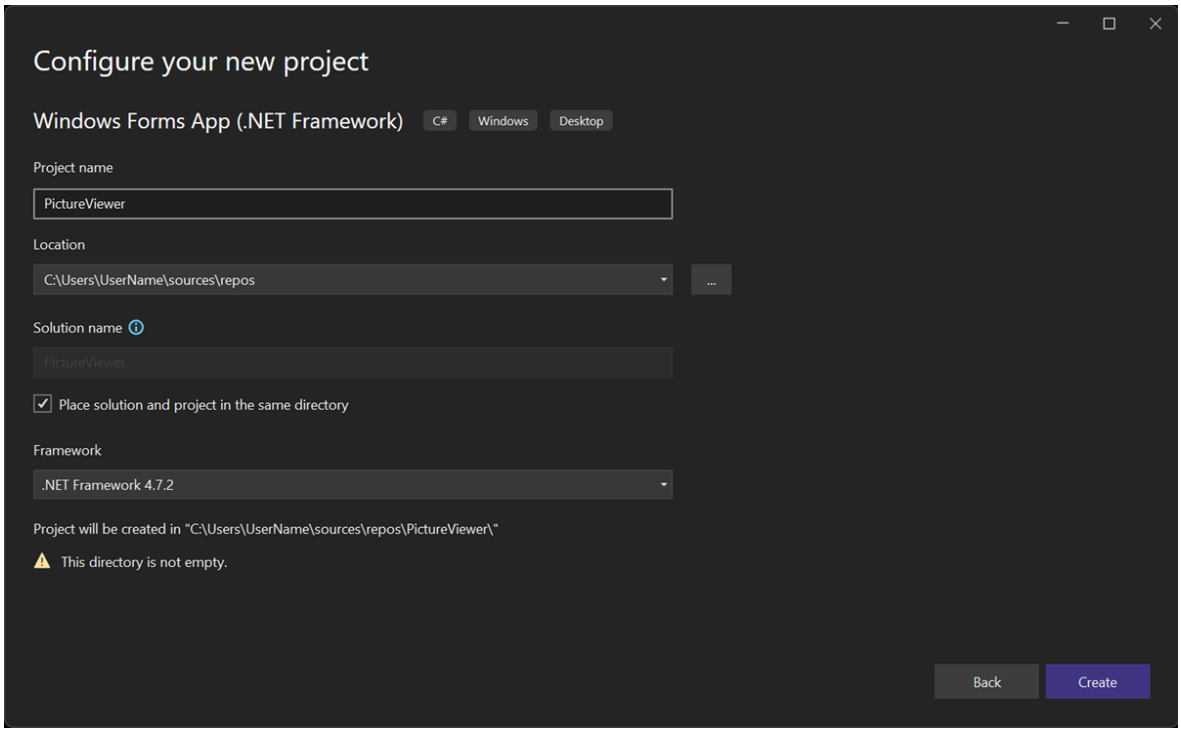
   随即打开“创建新项目”窗口，并显示几个项目模板。 模板包含给定项目类型所需的基本文件和设置。

3. 若要查找模板，可以在搜索框中键入或输入关键字。 系统会根据输入的关键字筛选可用模板列表。 可以通过从“所有语言”下拉列表中选择“C#”、从“所有平台”列表中选择“Windows”以及从“所有项目类型”列表中选择“控制台”进一步筛选模板结果。

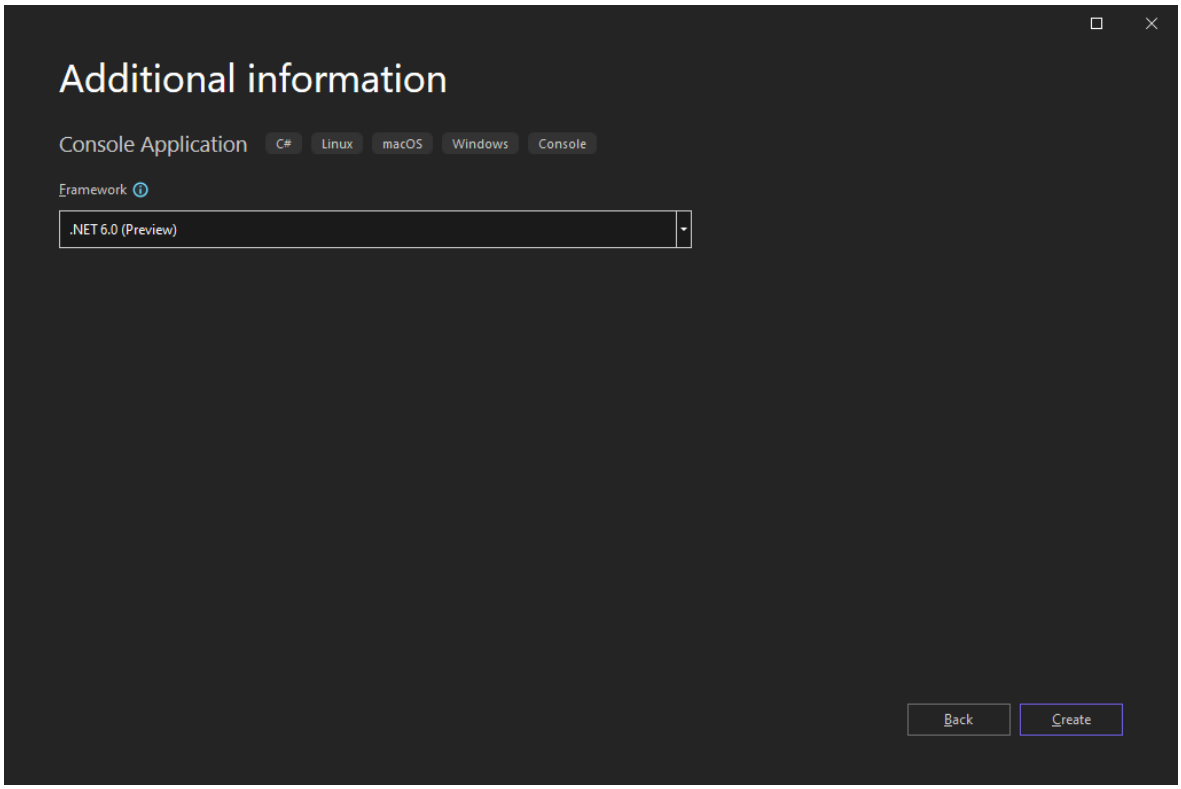
   选择“控制台应用程序”模板，然后选择“下一步”。



4. 在“配置新项目”窗口中，在“项目名称”框中输入“HelloWorld”。 （可选）更改项目目录的默认位置 C:\Users\<name>\source\repos，然后选择“下一步”。

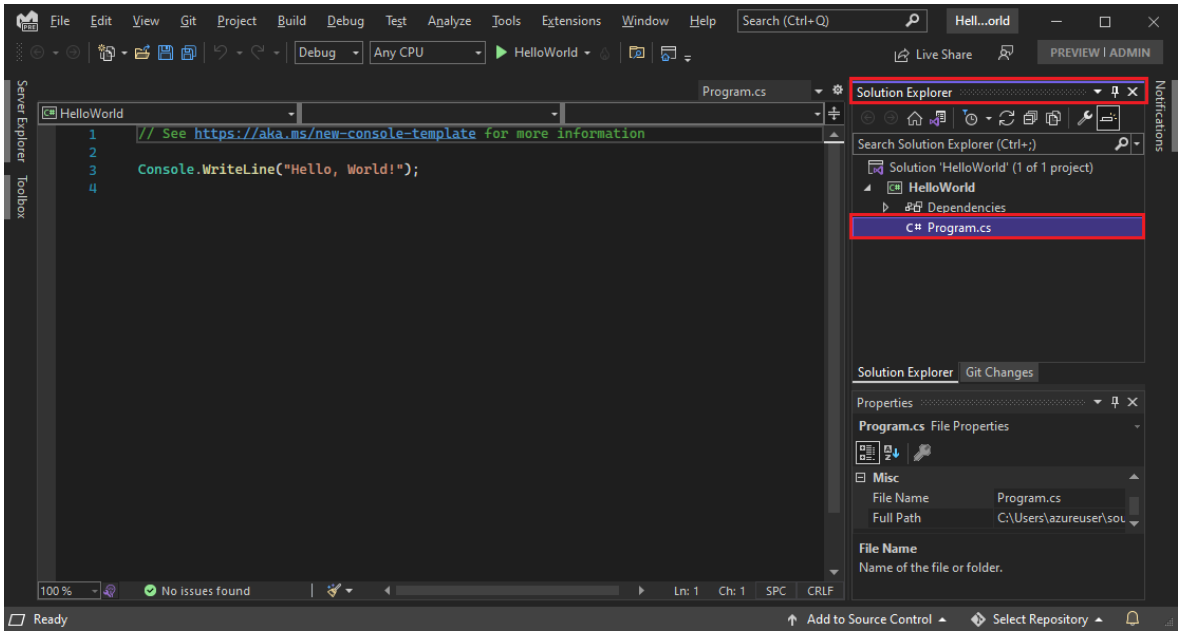


5. 在“附加信息”窗口中，验证“目标框架”下拉菜单中是否显示“.NET 6.0”，然后选择“创建”。



Visual Studio 随即创建项目。 该程序是简单的“Hello World”应用程序，可调用 Console.WriteLine() 方法在控制台窗口中显示字符串“Hello, World!”。

项目文件显示在名为“解决方案资源管理器”的窗口中 Visual Studio IDE 的右侧。 在“解决方案资源管理器”窗口中，选择“Program.cs”文件。 应用的 C# 代码在编辑器窗口中打开，会占用大部分空间。



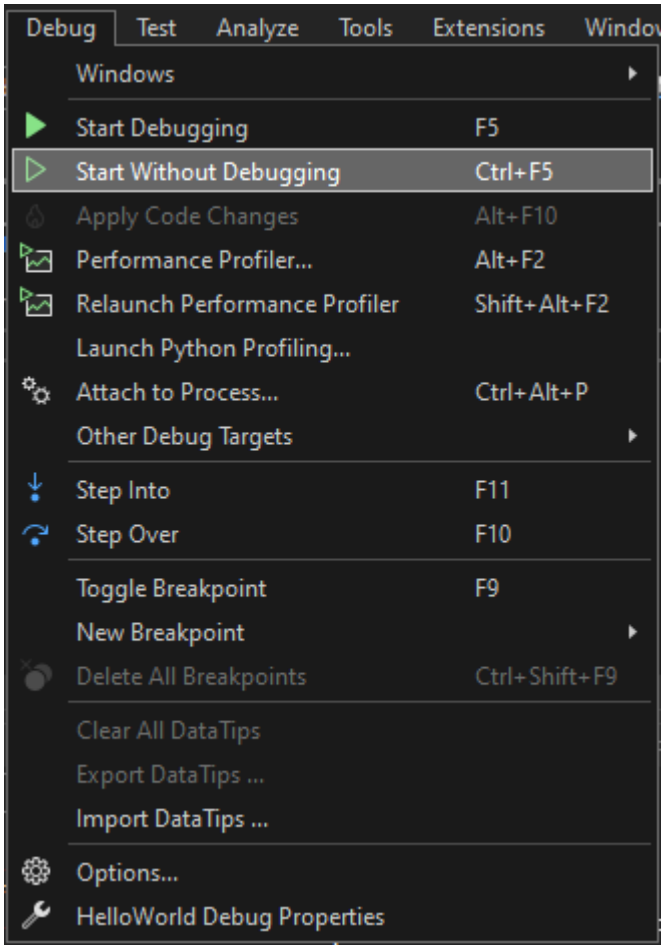
代码已自动着色，用于指示不同方面，如关键字或类型。 行号可帮助你查找代码。

代码中的小垂直虚线指示哪个大括号彼此匹配。 还可以通过选择带减号或加号的小方形来折叠或展开代码块。 此代码大纲功能可以隐藏不需要显示的代码，最大程度地减少屏幕混乱。

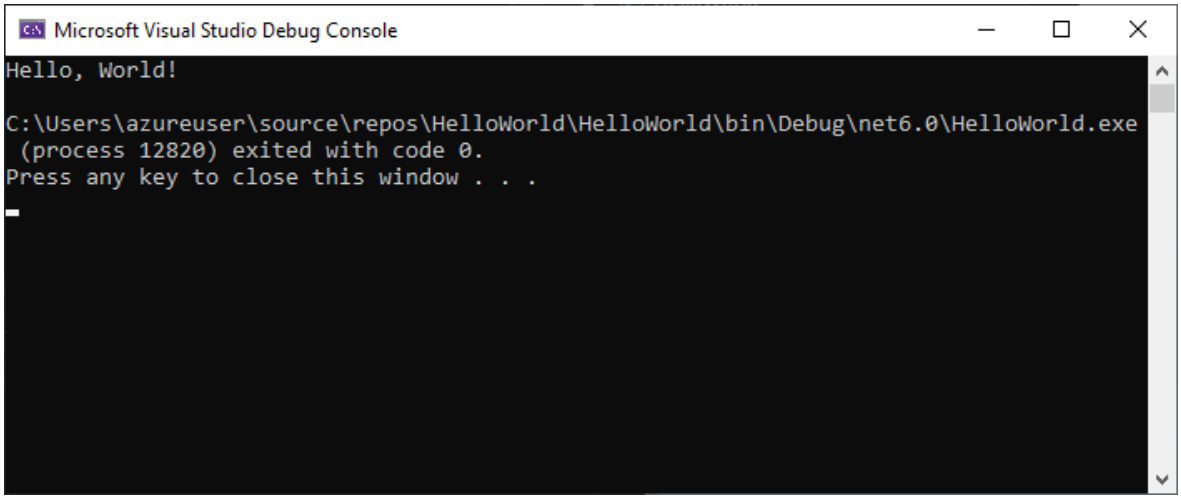
```
1   using ...
3
4   namespace CalculatorProgram
5   {
6
7       class Program
8       {
9           static void Main(string[] args)
10          {
11              bool endApp = false;
12              // Display title as the C# console calculator app.
13              Console.WriteLine("Console Calculator in C#\r");
14              Console.WriteLine("------------------------\n");
15
16              Calculator calculator = new Calculator();
17              while (!endApp)...
78              calculator.Finish();
79              return;
80
81
82          }
83      }
84
85          }
```

还提供了许多其他菜单和工具窗口。

6. 通过从 Visual Studio 顶部菜单中选择“调试”>“启动时不调用”来启动应用。 还可按 Ctrl+F5 。



Visual Studio 生成应用，控制台窗口随即打开并显示消息“Hello, World!”。 现在你拥有了一个正在运行的应用！



7. 按任意键关闭控制台窗口。

8. 接下来，向应用添加更多代码。 在 `Console.WriteLine("Hello World!");` 行的前面添加以下 C# 代码：

   ```
   C#

   Console.WriteLine("\nWhat is your name?");
   var name = Console.ReadLine();
   ```

   此代码在控制台窗口中显示“What is your name?”，然后等待用户输入文本。

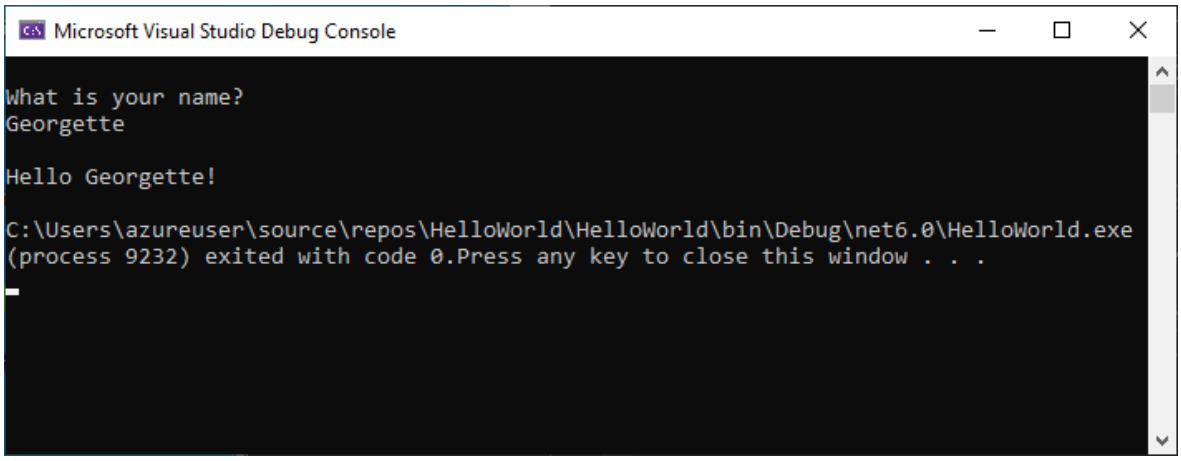
9. 将显示 `Console.WriteLine("Hello World!");` 的行更改为以下行：

   ```
   C#

   Console.WriteLine($"\nHello {name}!");
   ```

10. 选择“调试”>“开始执行(不调试)”，或按 Ctrl+F5，再次运行该应用。

    Visual Studio 重新生成应用，控制台窗口随即打开，并提示输入姓名。

11. 在控制台窗口中键入姓名，并按 Enter。



12. 按任意键关闭控制台窗口，并停止正在运行的程序。

使用重构和 IntelliSense

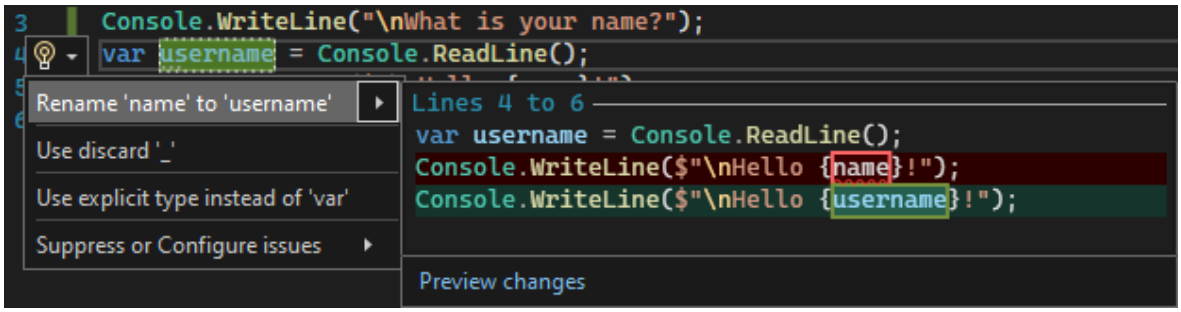
让我们了解一下如何借助重构和IntelliSense 更有效地进行编码。

首先，重命名 `name` 变量：

1. 双击 `name` 变量，然后键入变量“username”的新名称。

   变量周围将出现一个框，且边距中会出现灯泡。

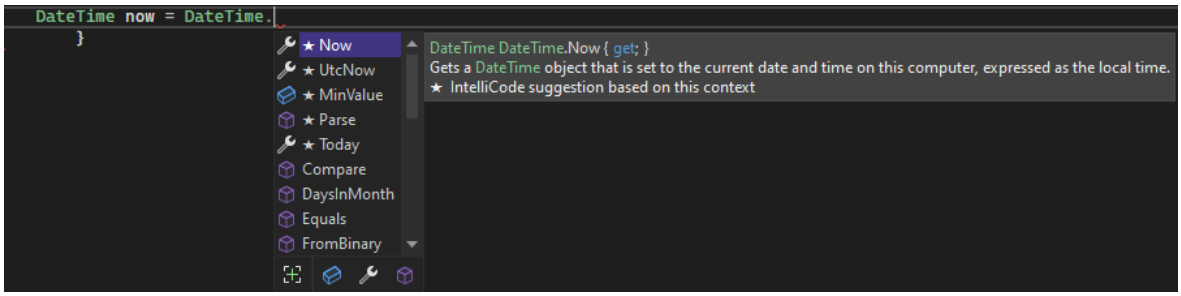
2. 选择灯泡图标，显示可用的快速操作。 选择“将 'name' 重命名为 'username'”。



   该变量会在整个项目中进行重命名，本例中只有两处。

3. 接下来了解下 IntelliSense。 在 `Console.WriteLine($"\nHello {username}!");` 行下方，键入 `DateTime now = DateTime.`。

   此时，框中显示 DateTime 类的成员。 当前所选成员的说明还会显示在单独的框中。



4. 通过双击或按 Tab 键，选择名为“Now”的成员，它是类属性。通过在行末尾添加分号来完成代码行：`DateTime now = DateTime.Now;`。

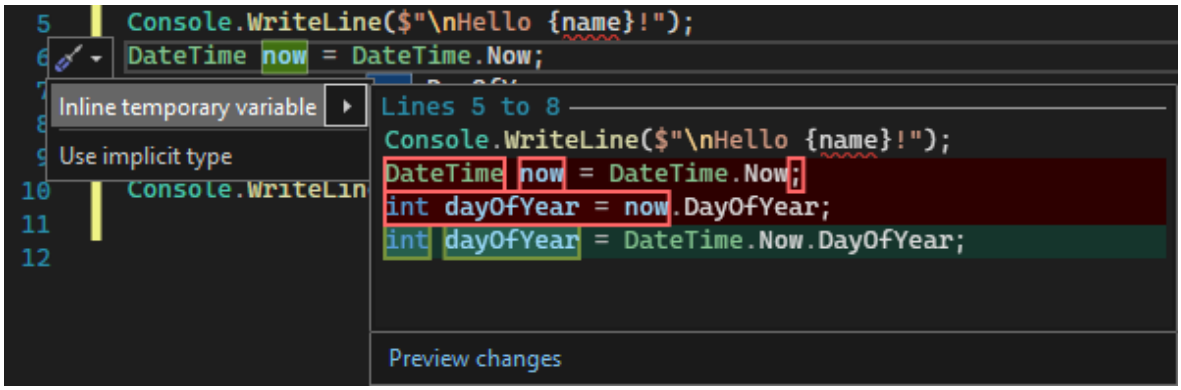
5. 在该行下，输入以下代码行：

```C#
int dayOfYear = now.DayOfYear;

Console.Write("Day of year: ");
Console.WriteLine(dayOfYear);
```

> **提示**
>
> **Console.Write** 与 **Console.WriteLine** 不同，它在打印后不会添加行终止符。 这意味着发送到输出的下一段文本将打印在同一行上。 将鼠标悬停在代码中的每个方法上，即可查看其说明。

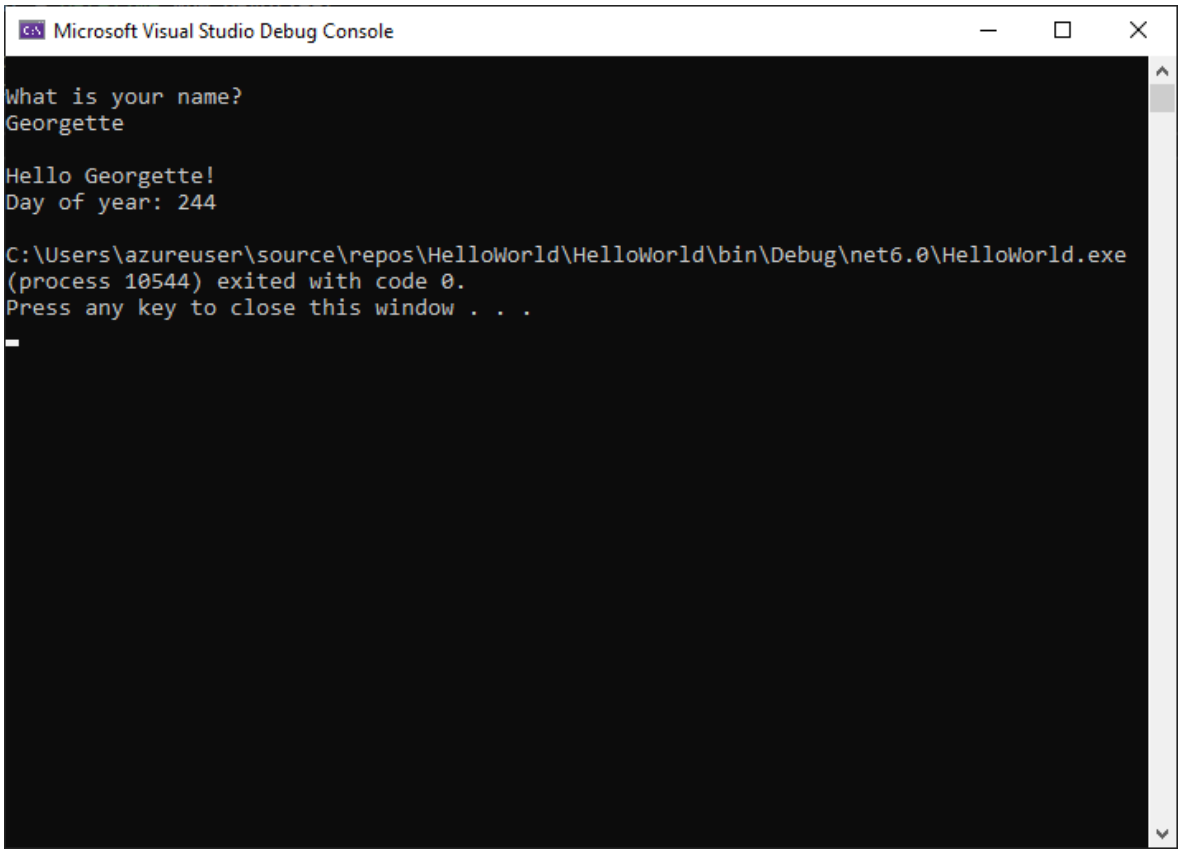
6. 接下来，再次使用重构来使代码更加简洁。 在行 `DateTime now = DateTime.Now;` 中选择变量 `now`。 该行的边距中会显示一个小螺丝刀图标。

7. 选择小螺丝刀图标以查看 Visual Studio 中的可用建议。 本例显示内联临时变量重构，可在无需更改整体代码行为的情况下删除代码行。



8. 选择“内联临时变量”以重构代码。

9. 按 Ctrl+F5 重新运行程序 。 输出的内容与以下类似：



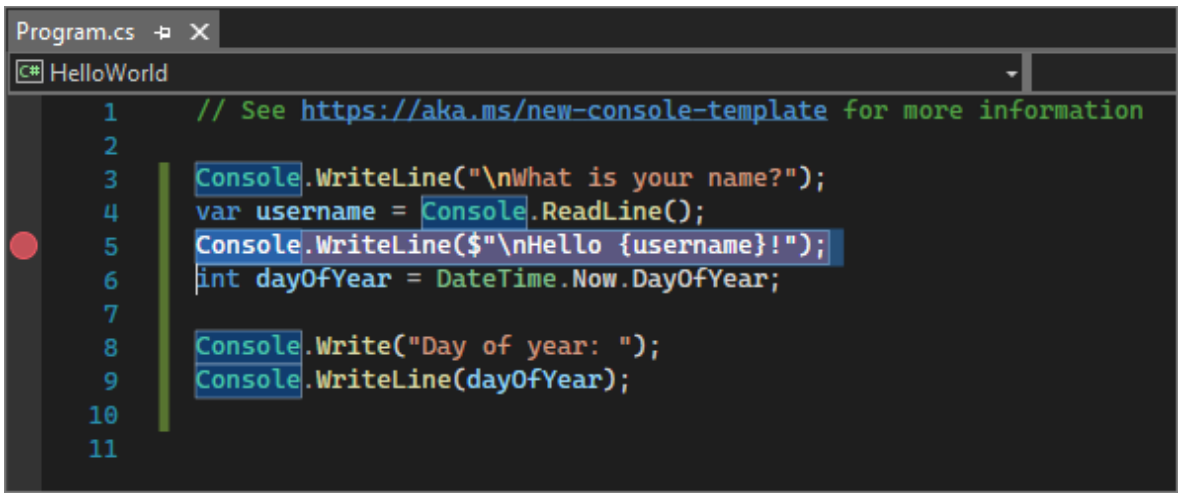
调试代码

编写代码时，应运行并测试该代码是否存在 bug。 可通过 Visual Studio 的调试系统逐句执行代码，一次执行一条语句，逐步检查变量。 可以设置在特定行停止代码执行的断点，并观察变量值在代码运行时的变化方式。

通过设置断点，可查看程序运行时 `username` 变量的值。

1. 通过单击代码行旁边最左边距（或装订线），在代码行上设置一个断点，该断点表示 `Console.WriteLine($"\nHello {username}!");`。 也可以选择代码行，然后按 F9。

   装订线中会出现一个红色圆圈，并突出显示该行。

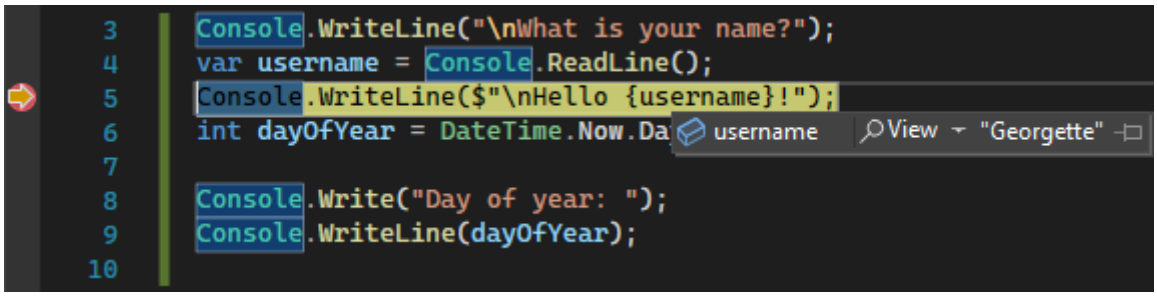


2. 选择“调试”>“启动调试”或按 F5，开始调试。

3. 控制台窗口出现并询问姓名时，请输入姓名。

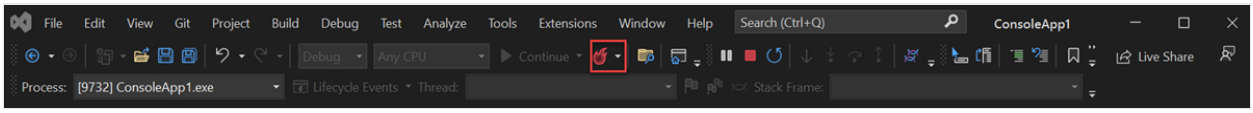
   Visual Studio 代码编辑器重新获得焦点，有断点的代码行突出显示为黄色。 黄色突出显示表示将在下一步执行此代码行。 断点使应用在此行暂停执行。

4. 将鼠标悬停在 `username` 变量上，即可查看它的值。 也可以右键单击 `username` 并选择“添加监视”，将变量添加到“监视”窗口，这样也可查看它的值。



5. 再次按 F5 以结束应用运行。

应用运行后，可通过单击“热重载”按钮将代码更改应用于正在运行的应用。



有关 Visual Studio 中调试的详细信息，请参阅调试器功能体验。

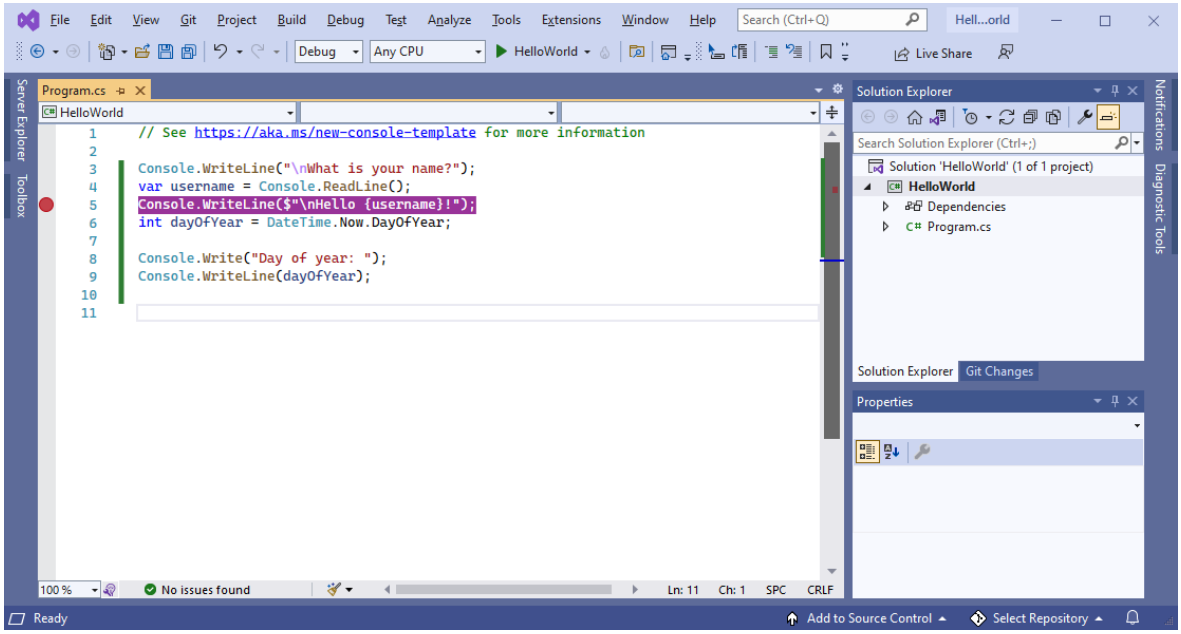
自定义 Visual Studio

可个性化设置 Visual Studio 用户界面，包括更改默认颜色主题。 若要更改颜色主题：

1. 在菜单栏中，选择“工具”>“选项”，打开“选项”对话框 。

2. 在“环境”>“常规”选项页上，将“颜色主题”选择内容更改为“蓝色”或“浅色”，然后选择“确定”。

   此时，整个 IDE 的颜色主题也会相应地更改。 以下屏幕截图显示“蓝色”主题：



若要了解有关 IDE 个性化设置的其他方法，请参阅个性化设置 Visual Studio。

选择环境设置

可以将 Visual Studio 配置为使用可满足 C# 开发人员需求的环境设置：

1. 在菜单栏上，选择“工具”>“导入和导出设置”。
2. 在“导入和导出设置向导”中，选择“重置所有设置”，然后选择“下一步”。
3. 在“保存当前设置”页上，选择是否在重置之前保存当前设置。 如果尚未自定义任何设置，请选择“不，只重置设置，同时覆盖我的当前设置”。 然后，选择“下一步”。
4. 在“选择默认设置集合”页上，依次选择“Visual C#”和“完成”。
5. 在“重置完成”页上，选择“关闭”。

若要了解有关 IDE 个性化设置的其他方法，请参阅个性化设置 Visual Studio。

后续步骤

查看下述一篇介绍性的文章，进一步了解 Visual Studio：

了解如何使用代码编辑器

了解项目和解决方案

请参阅

- 发现更多 Visual Studio 功能。
- 访问 visualstudio.microsoft.com 。
- 阅读 Visual Studio 博客 。

了解如何将代码编辑器用于 C#

项目 • 2023/06/19

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在这个 10 分钟的 Visual Studio 代码编辑器简介中，我们会向文件添加代码，了解 Visual Studio 编写、导航和了解 C# 代码的简便方法。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

本文假定你已熟悉 C#。 如果不熟悉，建议先看看 Visual Studio 中的 C# 和 ASP.NET Core 入门教程。

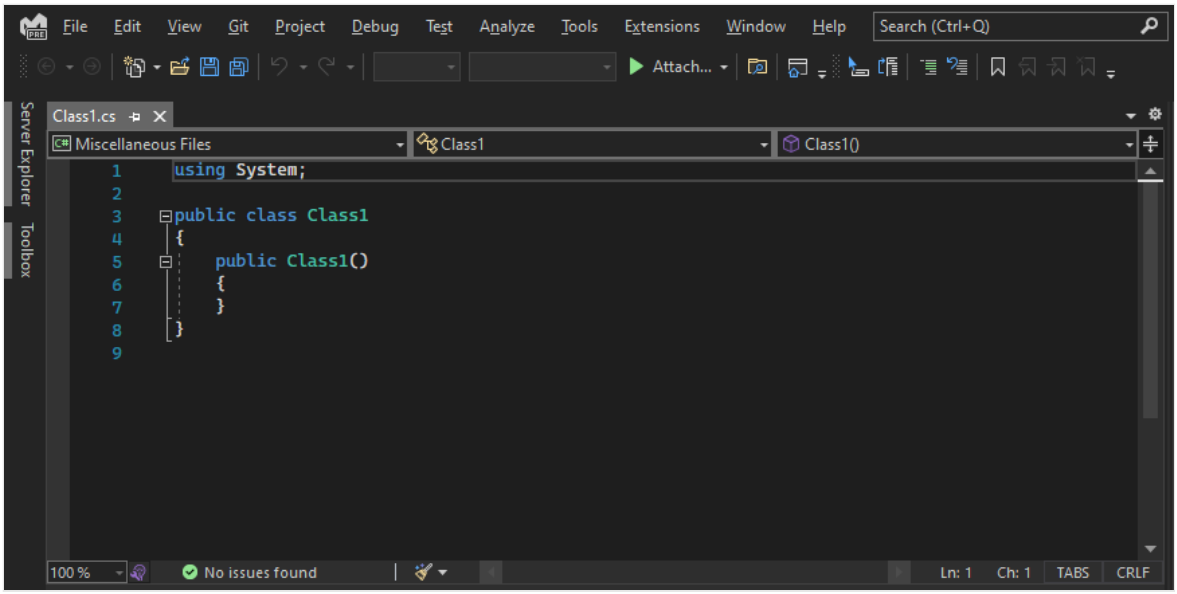
> **提示**
>
> 若要继续执行本文中的操作，请确保为 Visual Studio 选择了 C# 设置。 有关为集成开发环境 (IDE) 选择设置的信息，请参阅**选择环境设置**。

创建新代码文件

先创建一个新文件并向其添加一些代码。

1. 打开 Visual Studio。 按 Esc 或选择“开始”窗口中的“继续但无需代码”以打开开发环境。
2. 在菜单栏上的“文件”菜单中，选择“新建”>“文件”，或按 Ctrl+N 。
3. 在“新建文件”对话框的“常规”类别中，选择“Visual C# 类”，然后选择“打开” 。

   编辑器中将打开主干为 C# 类的新文件。 无需创建完整的 Visual Studio 项目来获取代码编辑器提供的某些益处，仅需一个代码文件即可。



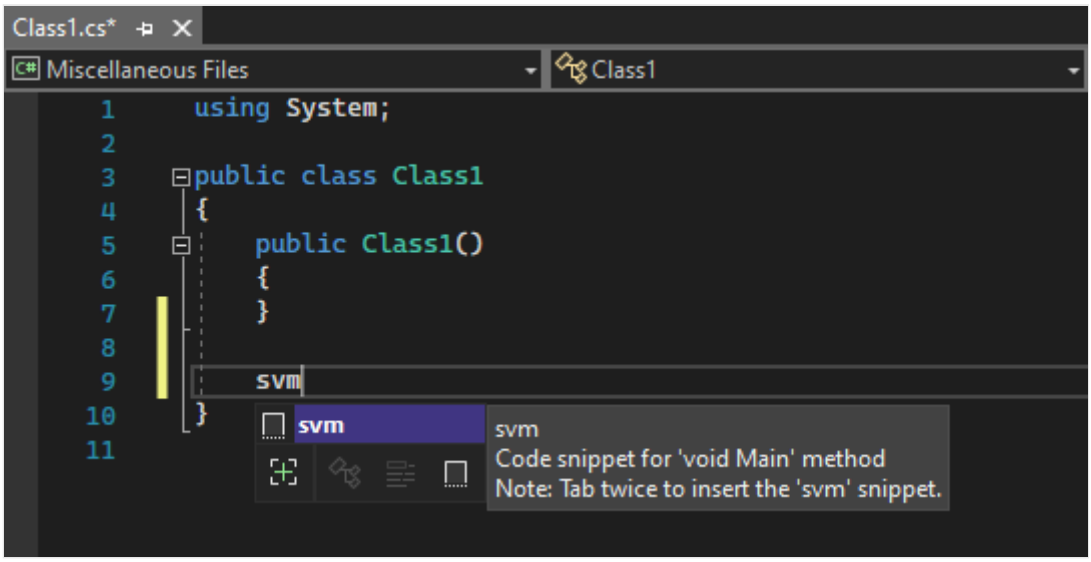
使用代码片段

Visual Studio 提供了实用的代码片段，可用于快速方便地生成常用代码块。 代码片段可用于不同编程语言，包括 C#、Visual Basic 和 C++。

我们将 C# `void Main` 代码片段添加到文件。

1. 将光标停在文件中最后的结束括号 `}` 的上方，并键入字符 `svm`。 `svm` 表示 `static void Main`，如果还不了解此含义，请不要担心。

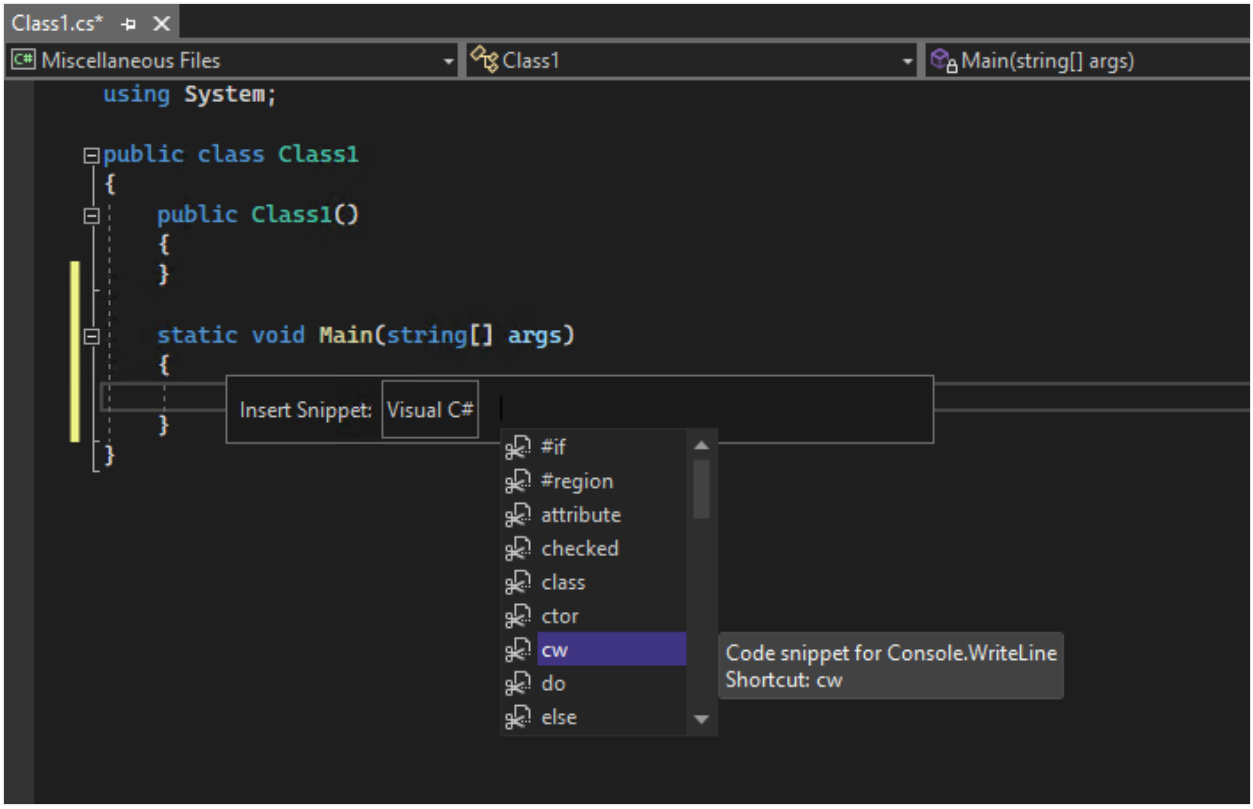
   随即将出现一个弹出对话框，其中包含有关 `svm` 代码片段的信息。



2. 按 Tab 两次，插入代码片段。

   你会看到 `static void Main()` 方法签名被添加到文件。 Main() 方法是 C# 应用程序的入口点。

对于不同编程语言，可用的代码片段不同。 依次选择“编辑”>“IntelliSense”>“插入代码片段”，或按 Ctrl+K、Ctrl+X，然后选择编程语言的文件夹，即可查看该语言的可用代码片段 。 对于 C#，代码片段列表如下所示：



该列表包含用于创建类、构造函数、for 循环、if 或 switch 语句等的代码片段。

为代码添加注释

工具栏是 Visual Studio 菜单栏下的一行按钮，有助于提高编码效率。 例如，可以切换到 IntelliSense 完成模式、增加或减少缩进，或者注释掉不想编译的代码。

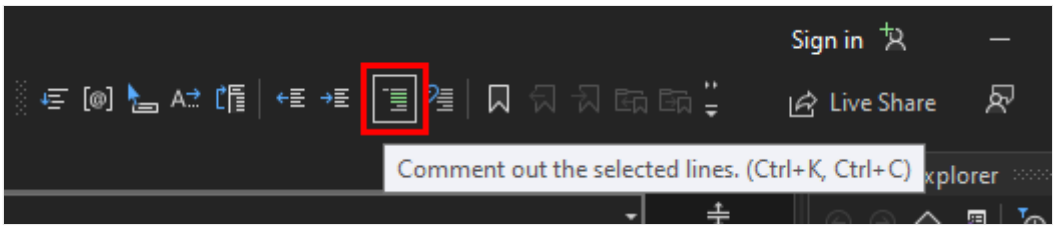
让我们注释掉一些代码。

1. 将以下代码粘贴到 `Main()` 方法主体中。

```C#
// someWords is a string array.
string[] someWords = {
    "the",
    "quick",
    "brown",
    "fox",
    "jumps"
```

```
};

string[] moreWords = {
    "over",
    "the",
    "lazy",
    "dog"
};

// Alphabetically sort the words.
IEnumerable<string> query = from word in someWords
                            orderby word
                            select word;
```

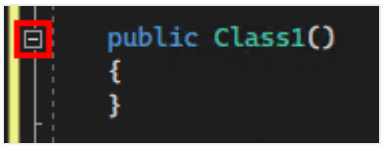
2. 我们现在没有使用 `moreWords` 变量，但稍后可能会用到，所以我们不想删除它。 那我们就来注释掉这些行。 选择整个 `moreWords` 定义直到结束分号，然后选择工具栏上的"注释掉选定行"。 如果想要使用键盘，请按 Ctrl+E, Ctrl+C 。



C# 注释字符 `//` 添加到了每个所选行的开始处，从而为代码添加注释。

折叠代码块

我们不想看到生成的 `Class1` 的空构造函数，所以为了让代码更整洁，我们将其折叠。 在构造函数第一行的边距中选择内部带有减号的小灰色框。 如果首选使用键盘，也可将光标置于构造函数代码中的任意位置，然后按 Ctrl+M、Ctrl+M 。



代码块折叠到第一行，后跟省略号 (`...`)。 若要再次展开代码块，请选择现在带有加号的相同灰色框，或者再次按 Ctrl+M，Ctrl+M 。 此功能被称为大纲显示，在折叠长方法或整个类时特别有用。

查看符号定义

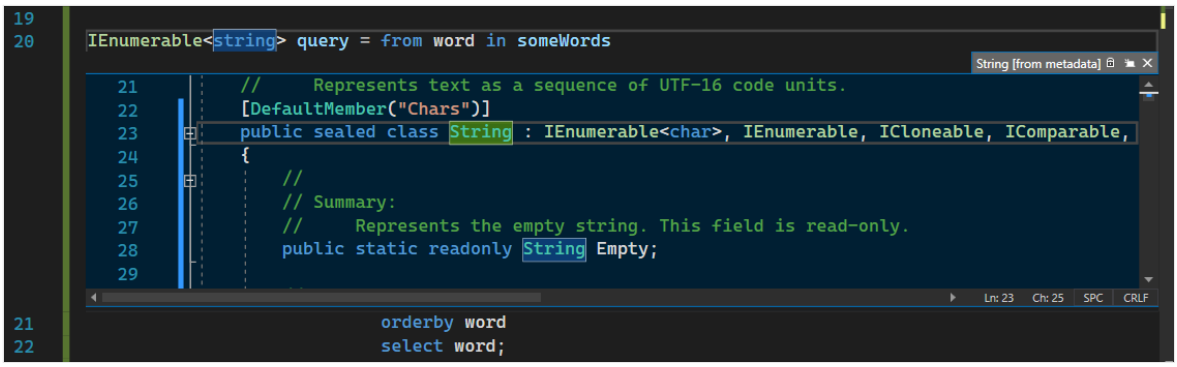
通过 Visual Studio 编辑器，可轻松查看类型、方法或变量的定义。 一种方法是转到包含定义的任何文件，例如通过选择"转到定义"或按"F12"，转到引用符号的任何位置 。 使用

“速览定义”速度更快，不会干扰你处理代码。

我们来快速查看一下 `string` 类型的定义。

1. 右键单击出现的任意 `string`，然后选择内容菜单上的“速览定义”。 或者，按 Alt+F12 。

   此时会出现一个弹出窗口，其中包含 `String` 类的定义。 可在弹出窗口中滚动，甚至还可从速览的代码中查看另一类型的定义。



2. 选择弹出窗口右上方的“x”小框，关闭“速览定义”窗口。

使用 IntelliSense 完成单词

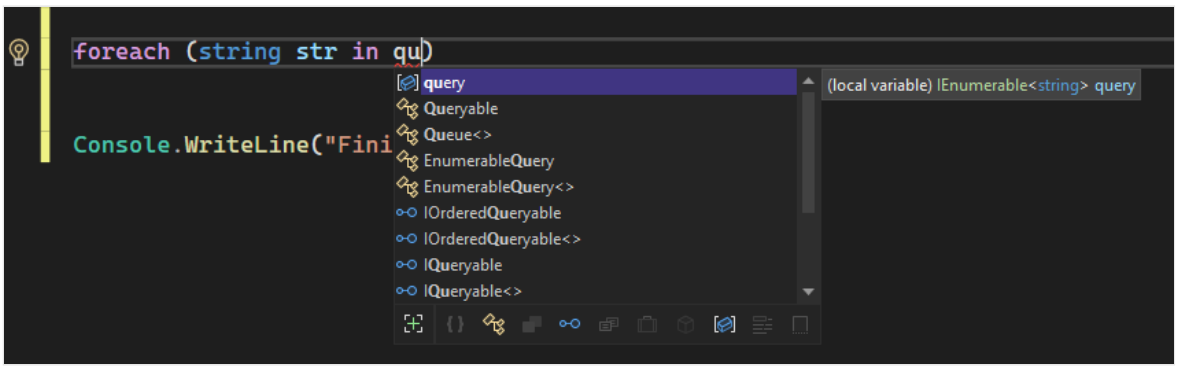
编写代码时，IntelliSense 是非常宝贵的资源。 它可显示某个类型的可用成员信息，或某个方法不同重载的参数详情。 还可用于完成单词，从而在输入大量字符后消除字符带来的歧义。

添加代码行，将有序字符串呈现到控制台窗口，这是程序输出的标准位置。

1. 在 `query` 变量下，开始键入以下代码：

```
C#

foreach (string str in qu
```

   此时会看到一个 IntelliSense 弹出项，其中包含有关 `query` 符号的相关信息。



2. 若要使用 IntelliSense 文字自动完成插入单词 `query` 的剩余部分，请按 Tab。

3. 完成后，代码块如以下代码所示。 你可以通过输入 `cw`，然后按 Tab 两次来生成 `Console.WriteLine` 语句，从而进一步练习代码片段。

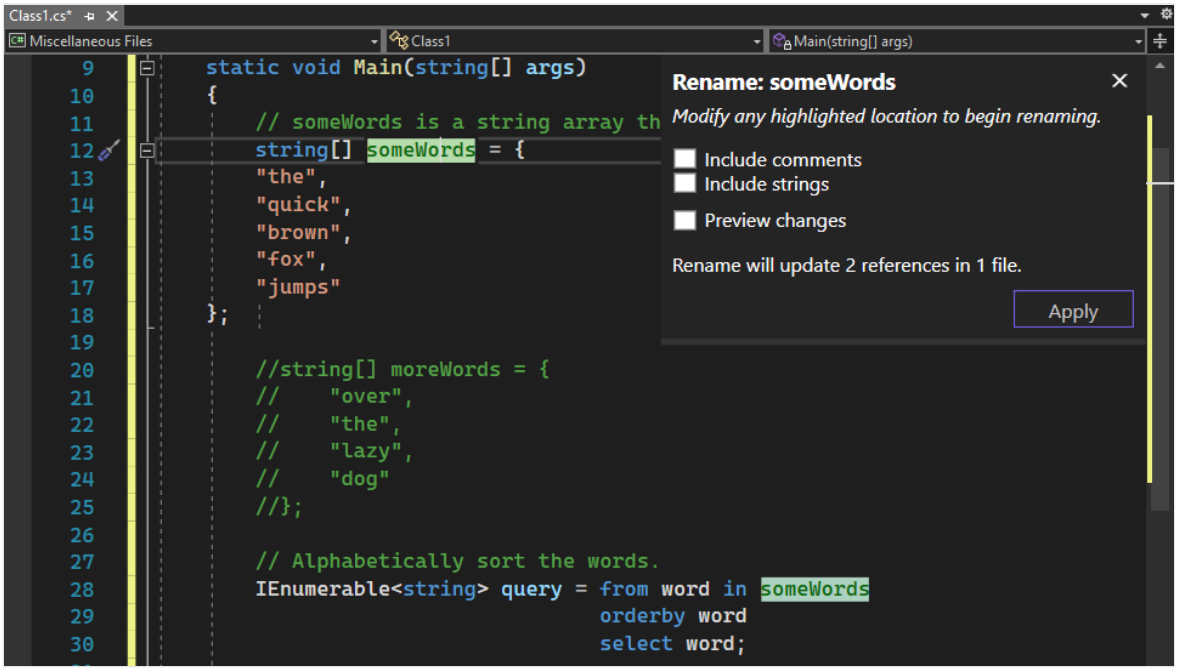
```
C#

foreach (string str in query)
{
   Console.WriteLine(str);
}
```

重构名称

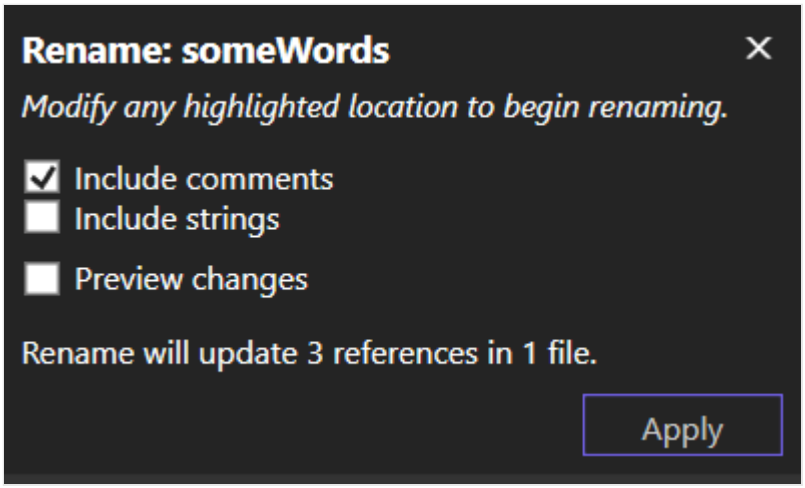
没有谁能一次就得到正确的代码，代码中可能必须要更改的一项内容是变量或方法的名称。 我们来试试 Visual Studio 的重构功能，将 `someWords` 变量重命名为 `unsortedWords`。

1. 将光标置于 `someWords` 变量的定义上，然后从右键菜单或上下文菜单中选择“重命名”，或按 F2。

   此时编辑器右上角会出现一个“重命名”对话框。



2. 输入所需名称“unsortedWords”。 你会看到查询中对 `query` 赋值语句中 `unsortedWords` 的引用也会自动重命名。 在按 Enter 前，请在“重命名”弹出框中选中“包含注释”复选框 。



3. 按 Enter，或在“重命名组”对话框中选择“应用”。

   代码中出现的两处 `someWords` 均被重命名，代码注释中的文本 `someWords` 也被重命名。

后续步骤

了解项目和解决方案

请参阅

- 代码片段
- 导航代码
- 大纲显示
- 转到定义和速览定义
- 重构
- 使用 IntelliSense

项目和解决方案简介

项目 • 2023/12/05

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在这篇介绍性的文章中，我们将探讨如何在 Visual Studio 中创建解决方案和项目 。 解决方案是一个容器，用于组织一个或多个相关的代码项目，例如，类库项目和对应的测试项目。

你将从头开始构建解决方案和项目，我们以此作为教学手段来帮助你理解项目的概念。 通常，你将使用 Visual Studio 项目模板来创建新项目。 你还将了解项目的属性及其包含的部分文件，并创建从一个项目到另一个项目的引用。

> **备注**
>
> 在 Visual Studio 中开发应用不需要用到解决方案和项目。 你可直接打开包含代码的文件夹，然后便可开始进行编码、生成和调试。 例如，克隆的 GitHub 存储库可能不包含 Visual Studio 项目和解决方案。 有关详细信息，请参阅**在 Visual Studio 中开发代码而无需创建项目或解决方案**。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

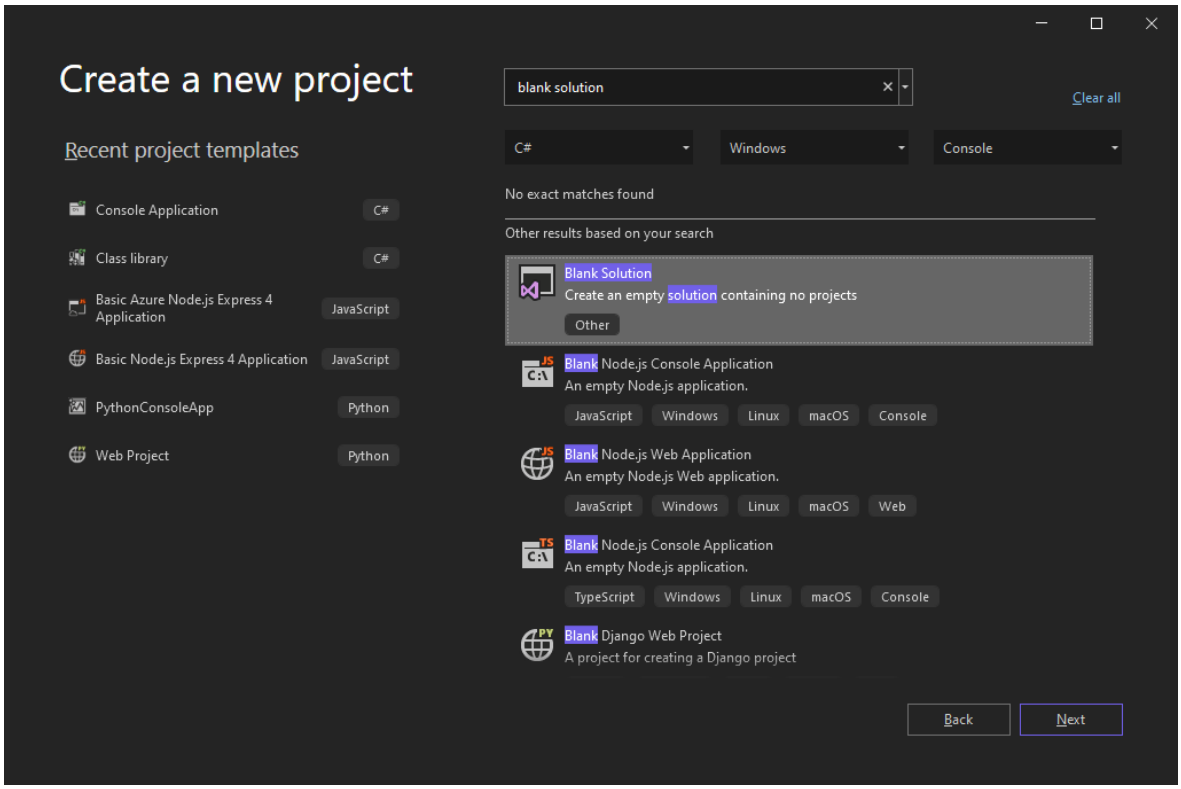
解决方案和项目

在 Visual Studio 中，解决方案不是“答案”。 解决方案仅仅是 Visual Studio 用来组织一个或多个相关项目的容器。 打开某个解决方案时，Visual Studio 会自动加载该解决方案包含的所有项目。

创建解决方案

首先创建一个空解决方案，以此来开始探索。 对 Visual Studio 有一定了解后，可能就不会经常创建空解决方案。 创建新项目时，Visual Studio 会自动为该项目创建一个解决方案，除非已经打开了一个解决方案。

1. 打开 Visual Studio，然后在“开始”窗口上，选择“创建新项目”。
2. 在“创建新项目”页面上的搜索框中键入“空白解决方案”，选择“空白解决方案”模板，然后选择“下一步” 。



> **提示**
>
> 如果你安装了多个工作负载，那么“空白解决方案”模板可能不会出现在搜索结果列表的顶部。 尝试滚动浏览“基于你搜索的其他结果”以查找模板。

3. 在“配置新项目”页面上，将解决方案命名为 QuickSolution，然后选择“创建” 。

   QuickSolution 解决方案随即出现在 Visual Studio 窗口右侧的“解决方案资源管理器”中 。 你将经常使用“解决方案资源管理器”来浏览项目的内容。

添加项目

现在，将你的第一个项目添加到解决方案。 先从空项目开始，添加所需的项。

1. 在“解决方案资源管理器”中右键单击“解决方案 'QuickSolution'”，然后从上下文菜单中选择“添加”>“新项目”。

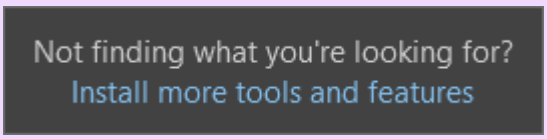
2. 在“添加新项目”页面顶部的搜索框中键入“空”，然后在“所有语言”下选择“C#” 。

3. 选择 C#“空项目(.NET Framework)”模板，然后选择“下一步” 。

> **备注**
>
> Visual Studio 使用基于工作负载的安装旨在仅安装所执行的开发类型需要的组件。 如果没有看到“空项目(.NET Framework)”模板，则需要安装“.NET 桌面开

发”Visual Studio 工作负载 。

在创建新项目时安装新工作负载的简便方法是，在显示“未找到你要查找的内容”的文本下选择“安装更多工具和功能”链接 。 在 Visual Studio 安装程序中，选择“.NET 桌面开发”工作负载，然后选择“修改” 。



4. 在“配置新项目”页面中，将项目命名为 QuickDate，然后选择“创建” 。

   QuickDate 项目随即出现在“解决方案资源管理器”中的解决方案下 。 项目包含一个引用节点和名为“App.config”的单个文件。

向项目添加一个项

将代码文件添加到你的空项目。

1. 在“解决方案资源管理器”中的“QuickDate”项目的右键菜单或上下文菜单中，依次选择“添加”>“新建项” 。

   此时将打开“添加新项”对话框。 如果对话框以紧凑视图打开，请选择“显示所有模板”。

2. 展开“Visual C# 项”，然后选择“代码” 。 在中间窗格中，选择“类”项模板。 在“名称”下，键入 Calendar，然后选择“添加”。

   Visual Studio 随即将名为 Calendar.cs 的文件添加到项目。 末尾的 .cs 是 C# 代码文件的文件扩展名。 Calendar.cs 文件出现在“解决方案资源管理器”的直观项目层次结构中，文件在编辑器中打开 。

3. 将 Calendar.cs 文件的内容替换为以下代码：

```
C#

using System;

namespace QuickDate
{
    internal class Calendar
    {
        static void Main(string[] args)
        {
            DateTime now = GetCurrentDate();
            Console.WriteLine($"Today's date is {now}");
```

```
            Console.ReadLine();
        }

        internal static DateTime GetCurrentDate()
        {
            return DateTime.Now.Date;
        }
    }
}
```

你还不需要了解代码执行的所有操作。 按 Ctrl+F5 运行应用，查看应用是否将今天的日期输出到控制台（标准输出）窗口 。 然后，关闭控制台窗口。

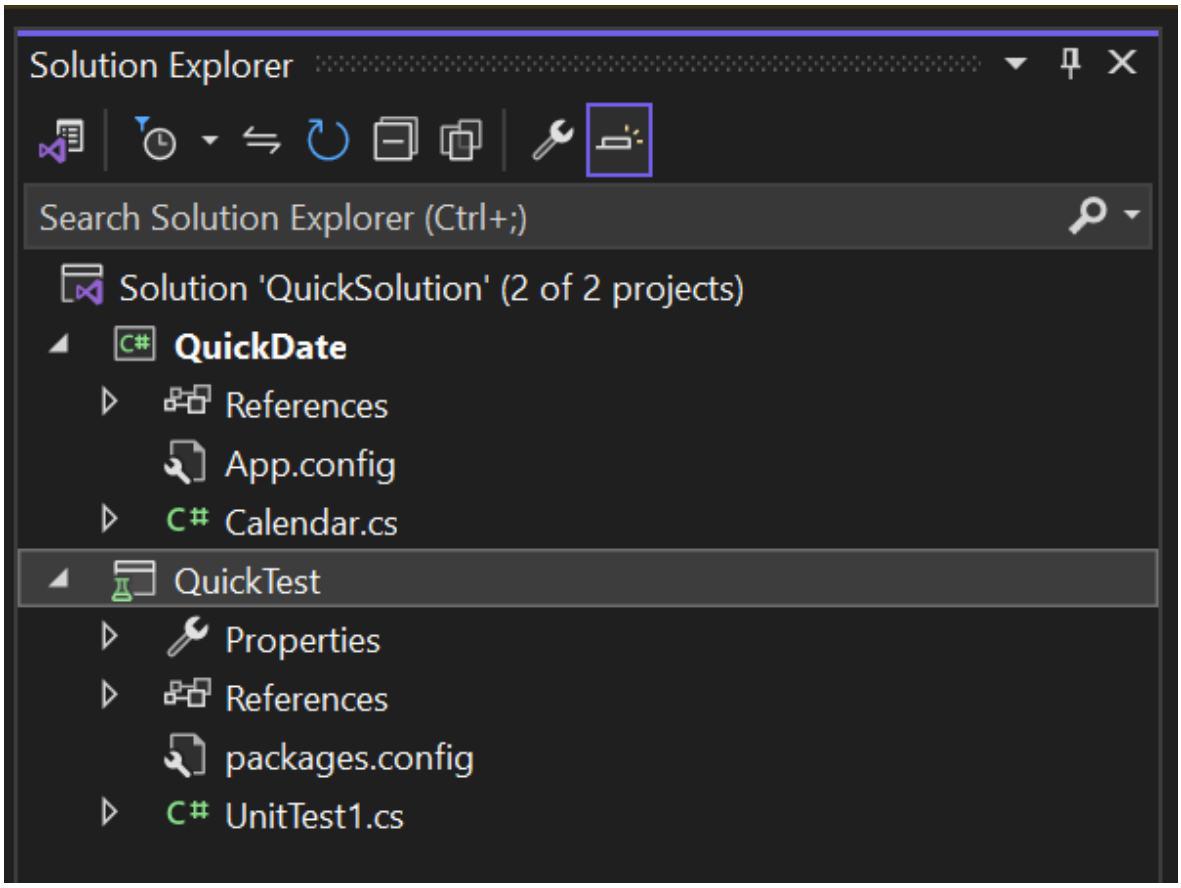
添加第二个项目

解决方案通常包含多个项目，这些项目通常相互引用。 解决方案中的一些项目可能是类库，一些可能是可执行应用程序，一些可能是单元测试项目或网站。

要将单元测试项目添加到解决方案，请从项目模板开始，这样就无需将其他代码文件添加到项目。

1. 在“解决方案资源管理器”中的“解决方案 'QuickSolution'”的右键菜单或上下文菜单中，依次选择“添加”>“新建项目” 。
2. 在“添加新项目”对话框顶部的搜索框中键入“单元测试”，然后在“语言”下选择“C#” 。
3. 选择 C#“单元测试项目(.NET Framework)”项目模板，然后选择“下一步” 。
4. 在“配置新项目”页面中，将项目命名为 QuickTest，然后选择“创建”。

   Visual Studio 随即将 QuickTest 项目添加到“解决方案资源管理器”，并在编辑器中打开 UnitTest1.cs 文件 。



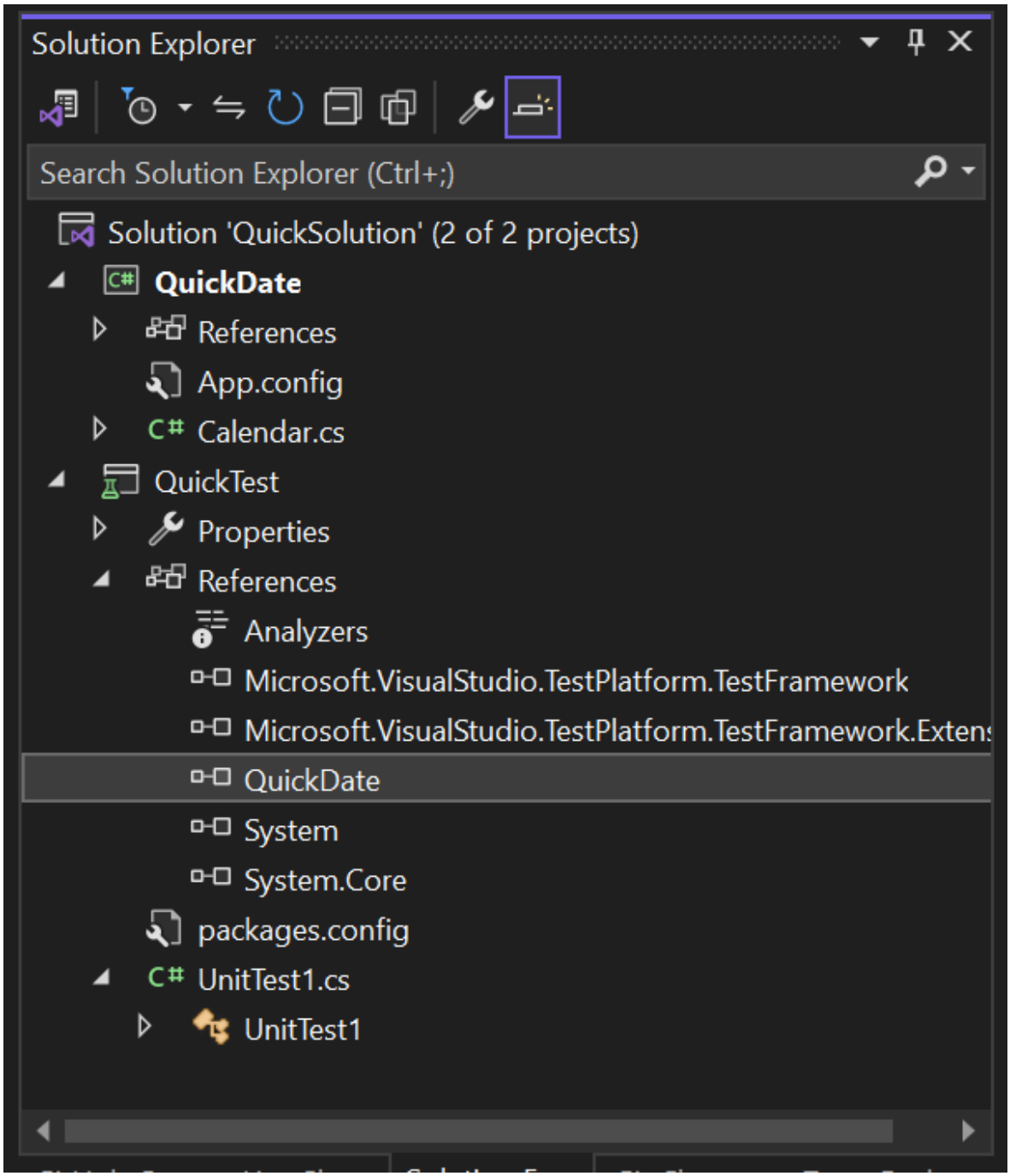
添加项目引用

你将使用新的单元测试项目测试 QuickDate 项目中的方法，因此需要将对 QuickDate 的引用添加到 QuickTest 项目 。 添加引用会在两个项目之间创建生成依赖关系，这意味着生成解决方案时，会先生成 QuickDate，再生成 QuickTest 。

1. 在“解决方案资源管理器”中，右键单击 QuickTest 项目的“引用”节点，然后从上下文菜单中选择“添加引用” 。

2. 在“引用管理器”对话框中，选择“项目”。 在中间窗格中，选择“QuickDate”旁的复选框，然后选择“确定” 。

   对 QuickDate 项目的引用随即出现在“解决方案资源管理器”中的 QuickTest 项目下 。



添加测试代码

1. 现在，将测试代码添加到 C# 测试代码文件。 将 *UnitTest1.cs* 的内容替换为以下代码：

```C#
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace QuickTest
{
    [TestClass]
    public class UnitTest1
    {
```

```
        [TestMethod]
        public void TestGetCurrentDate()
        {
            Assert.AreEqual(DateTime.Now.Date,
QuickDate.Calendar.GetCurrentDate());
        }
    }
}
```

一些代码下方会出现红色波浪线。 你可以通过将测试项目设为 QuickDate 项目的友元程序集来解决此错误。

2. 在 Calendar.cs 文件中，将以下 using 语句和 InternalsVisibleToAttribute 属性添加到文件顶部，以解决测试项目中的错误。

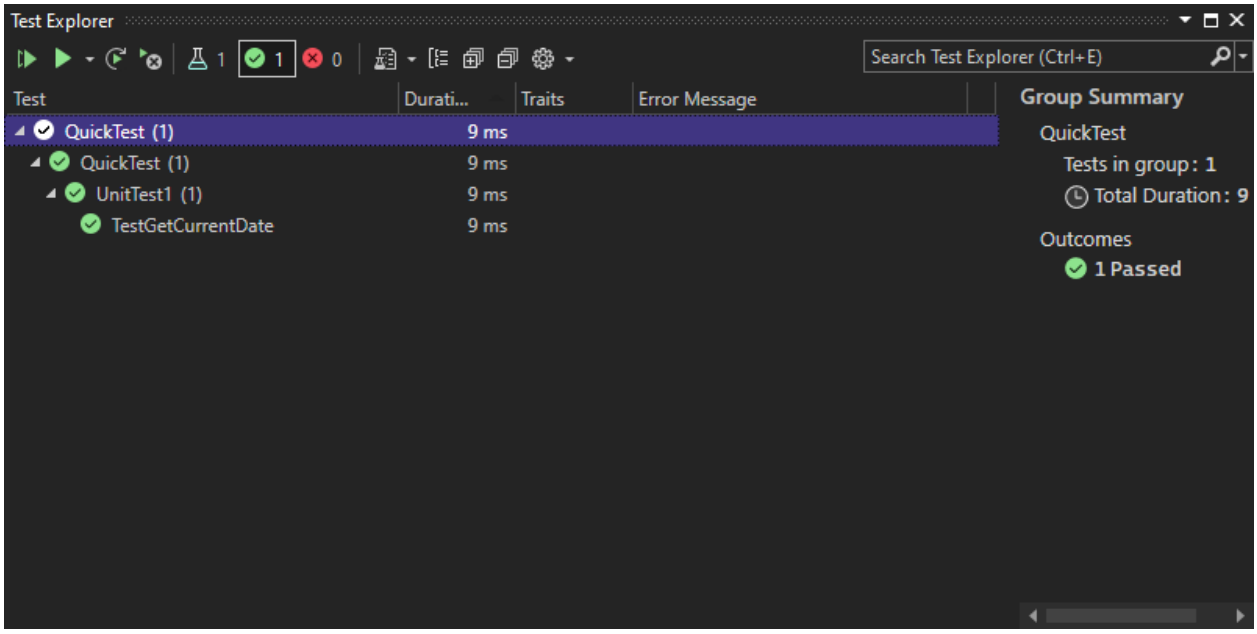
```
C#

using System.Runtime.CompilerServices;

[assembly: InternalsVisibleTo("QuickTest")]
```

Calendar.cs 代码应类似于如下屏幕截图：

```
1  using System;
2  using System.Runtime.CompilerServices;
3
4
5   [assembly: InternalsVisibleTo("QuickTest")]
6
7
8  namespace QuickDate
9   {
        1 reference
10      internal class Calendar
11      {
            0 references
12          static void Main(string[] args)
13          {
14              DateTime now = GetCurrentDate();
15              Console.WriteLine($"Today's date is {now}");
16              Console.ReadLine();
17          }
18
            2 references
19          internal static DateTime GetCurrentDate()
20          {
21              return DateTime.Now.Date;
22          }
23      }
24  }
```

运行单元测试

要检查单元测试是否正常工作，请从菜单栏中依次选择“测试”>“运行所有测试”。 “测试资源管理器”窗口随即打开，你应该会看到 TestGetCurrentDate 测试通过。



> **提示**
>
> 还可以在菜单栏中选择“测试”>“测试资源管理器”，以打开“测试资源管理器”。

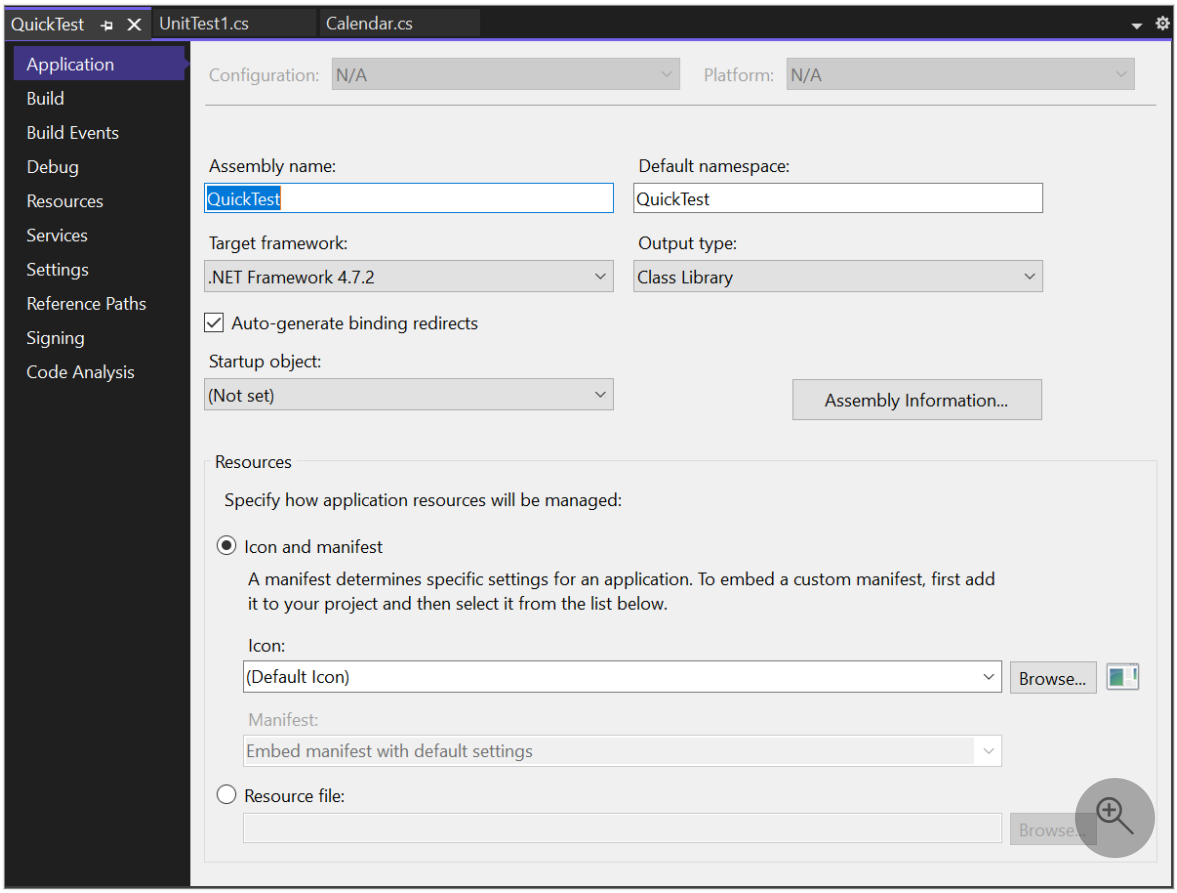
项目属性

包含 InternalsVisibleToAttribute 属性的 Calendar.cs 文件中的行引用 QuickTest 项目的程序集名称或文件名。 程序集名称可能不会始终与项目名称相同。 要查找项目的程序集名称，请使用项目属性。 属性页包含项目的各种设置。

1. 在“解决方案资源管理器”中，右键单击 QuickTest 项目，然后选择“属性”，或者选择该项目，然后按 Alt+Enter。

   项目的属性页打开到“应用程序”选项卡。QuickTest 项目的程序集名称确实是 QuickTest。

   如果需要，可以在此处更改该名称。 随后，在生成测试项目时，生成的二进制文件的名称将从 QuickTest.dll 更改为 <NewName>.dll。



2. 了解项目属性页的其他选项卡，例如“生成”和“调试”。 这些选项卡对不同类型的项目是不同的。

另请参阅

- 使用项目和解决方案
- 在 Visual Studio 中开发代码而无需创建项目或解决方案
- 管理项目和解决方案属性
- 管理项目中的引用

Visual Studio 的功能

项目 • 2023/09/03

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

本文介绍的功能适合有经验的开发人员或已熟悉 Visual Studio 的开发人员。 有关 Visual Studio 的基本简介，请参阅 Visual Studio IDE 概述。

模块化安装

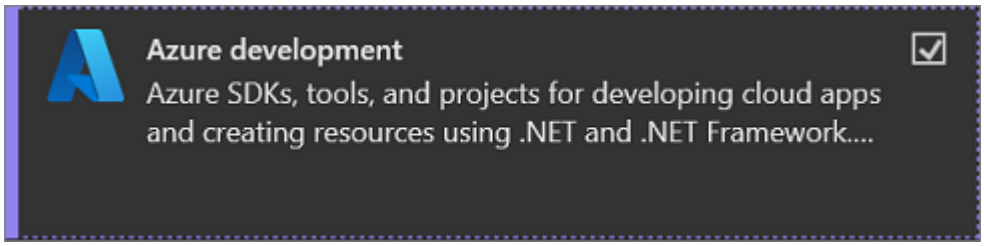
在 Visual Studio 的模块化安装程序中，选择和安装所需的工作负载。 工作负载是编程语言或平台所需的功能组。 此模块化策略使安装 Visual Studio 占用的空间更小，因此安装和更新速度更快。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

要了解设置系统上的 Visual Studio 的详细信息，请参阅安装 Visual Studio。

创建已启用云的 Azure 应用

Visual Studio 具有可用于轻松地创建已启用 Microsoft Azure 云的应用程序的工具套件。 可以从 Visual Studio 集成开发环境 (IDE) 直接配置、生成、调试、打包和部署 Azure 应用和服务。 若要获取 Azure 工具和项目模板，安装 Visual Studio 时请选择“Azure 开发” 工作负载。



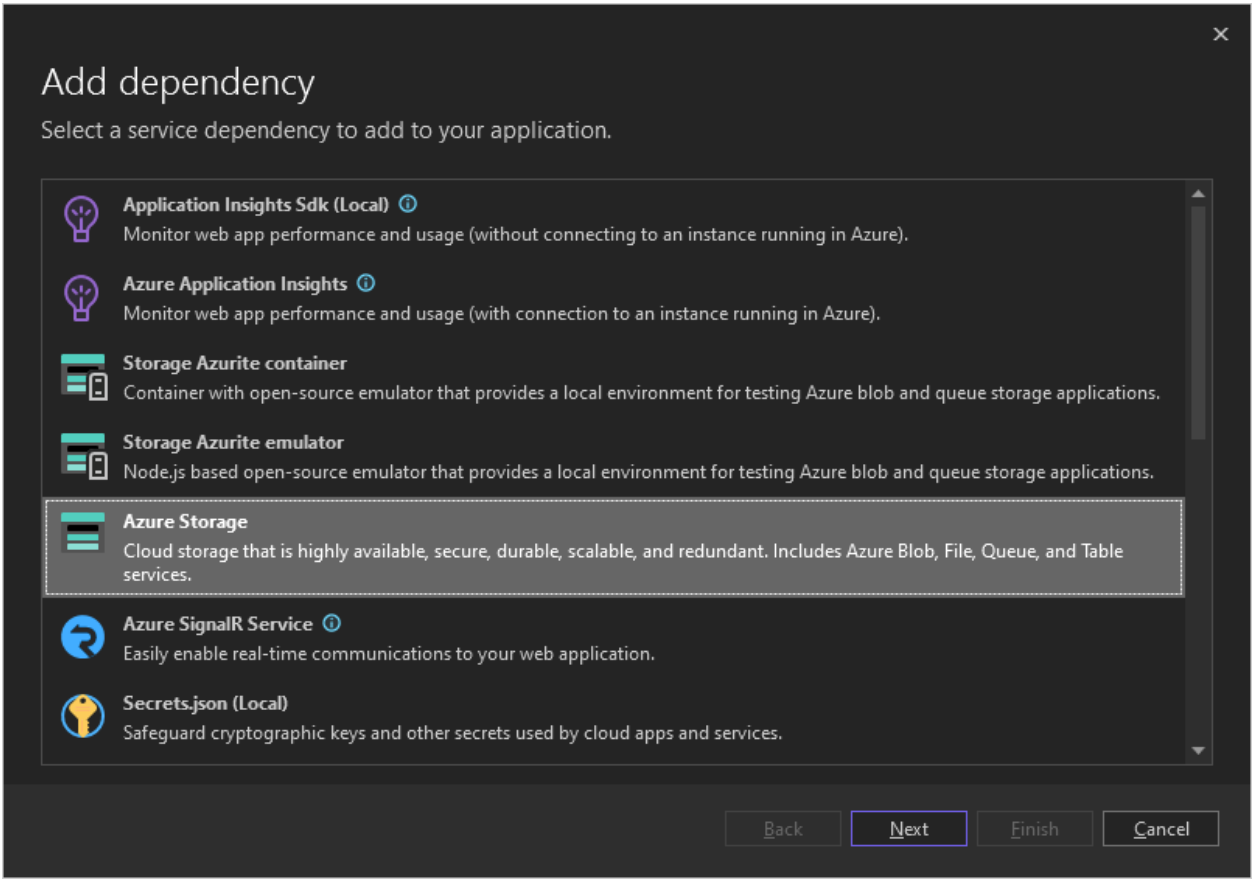
> **ⓘ 重要**
>
> “Cloud Explorer”窗口已在 Visual Studio 2022 中停用。 有关详细信息，请参阅**在 Visual Studio Cloud Explorer 中管理与 Azure 帐户关联的资源**。
>
> 使用 Azure 门户根据需要访问 Azure 资源。 可以继续使用早期版本的 Visual Studio 中服务器资源管理器的 Azure 节点。

可通过添加以下连接服务为应用使用 Azure 服务：

- Active Directory 连接服务，通过 Azure Active Directory (Azure AD) 帐户连接到 Web 应用
- Azure 存储连接服务，该服务适用于 blob 存储、队列和表
- 密钥保管库连接服务，可用于管理 Web 应用的机密

项目类型决定了可用的连接服务 。 右键单击“解决方案资源管理器”中的项目并选择“添加”>“连接服务”来添加服务。

在“连接服务”屏幕上，选择链接或加号以添加服务依赖关系。 在“添加依赖关系”屏幕上，选择要添加的服务，然后按照屏幕上的提示连接到 Azure 订阅和服务。



有关详细信息，请参阅使用 Visual Studio 和 Azure 移动到云。

创建 Web 应用

Visual Studio 可以帮助你编写 Web 应用。 可以使用 ASP.NET、Node.js、Python、JavaScript 和 TypeScript 来创建 Web 应用。 Visual Studio 支持多种 Web 框架，如 Angular、jQuery 和 Express。

ASP.NET Core 和 .NET Core 在 Windows、Mac 和 Linux 操作系统上运行。 ASP.NET Core 是 MVC、WebAPI 和 SignalR 的一个重大更新。 ASP.NET Core 旨在完全提供可组合的精益 .NET 堆栈，以便生成基于云的新式 Web 应用和服务。

有关详细信息，请参阅新式 Web 工具。

生成跨平台应用和游戏

Visual Studio 可以生成适用于 macOS、Linux 和 Windows，以及 Android、iOS 和其他移动设备 的应用和游戏。 使用 Visual Studio 可以：

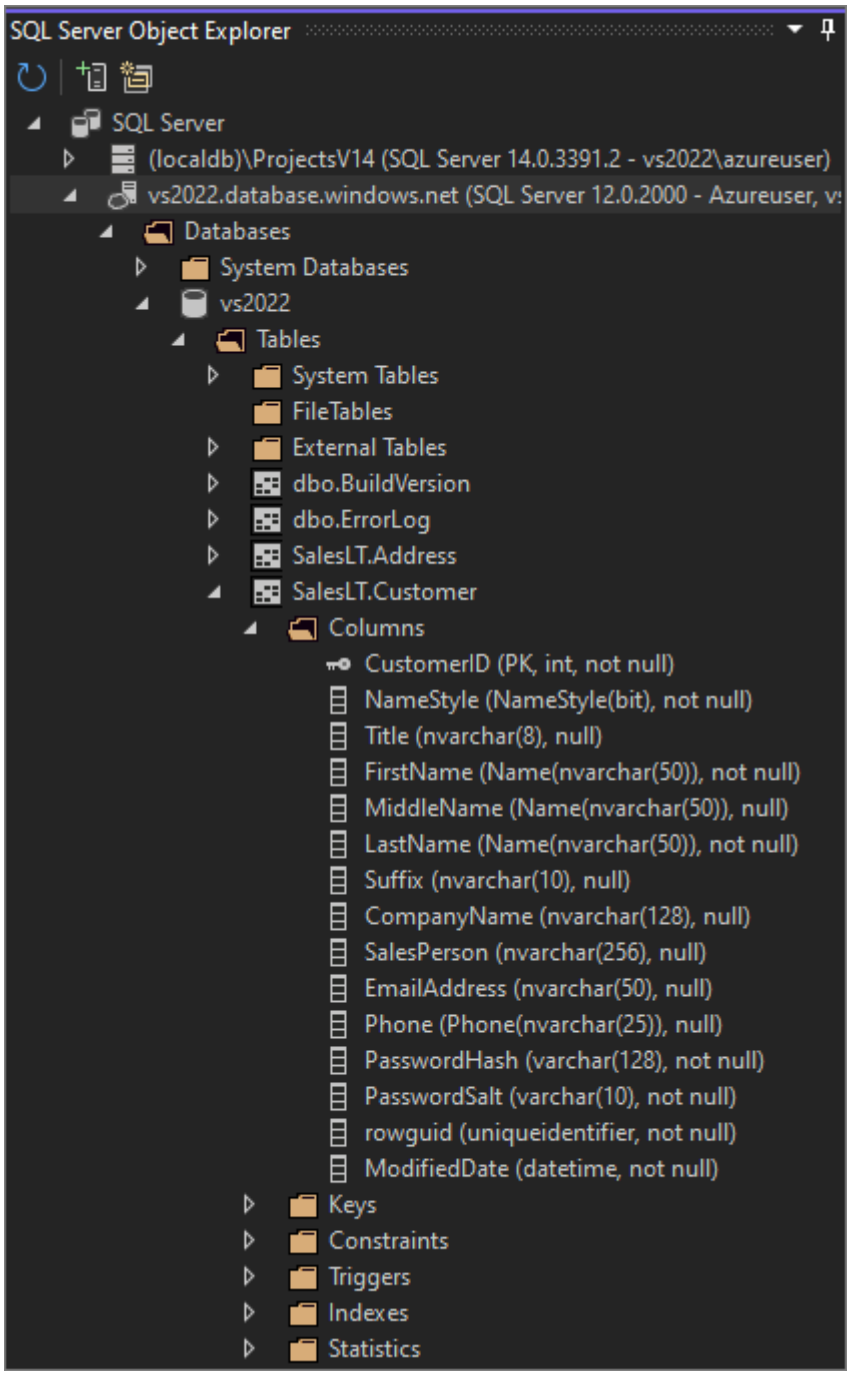
- 生成在 Windows、macOS 和 Linux 上运行的 .NET Core 应用。
- 通过使用 Xamarin，在 C# 和 F# 中生成适用于 iOS、Android 和 Windows 的移动应用。
- 通过使用 Visual Studio Tools for Unity，在 C# 中生成 2D 和 3D 游戏。
- 生成适用于 iOS、Android 和 Windows 设备的本机 C++ 应用。 通过适用于跨平台开发的 C++，在 iOS、Android 和 Windows 库中分享通用代码。

连接到数据库

服务器资源管理器有助于你浏览和管理本地、远程以及 Azure、Microsoft 365、Salesforce.com 和网站上的服务器实例及资产。 若要打开“服务器资源管理器”，请选择“视图”>“服务器资源管理器”。 有关使用服务器资源管理器的详细信息，请参阅添加新连接。

SQL Server 对象资源管理器提供类似于 SQL Server Management Studio 中的数据库对象。 使用 SQL Server 对象资源管理器可以执行轻负载数据库的管理和设计工作。 示例包括使用上下文菜单编辑表数据、对比架构和执行查询等等。

若要打开“SQL Server 对象资源管理器”，请选择“服务器资源管理器”窗口顶部的图标，或从 Visual Studio 顶部菜单中选择“视图”>“SQL Server 对象资源管理器”。



SQL Server Data Tools (SSDT) 是一个适用于 SQL Server、Azure SQL 数据库和 Azure SQL 数据仓库的强大的开发环境。 通过 SSDT 可以生成、调试、维护和重构数据库。 可使用数据库项目，或直接使用已连接的数据库实例（本地或非本地）。 若要获取 SSDT，请使用 Visual Studio 安装程序安装“数据存储和处理”工作负载。

调试、测试和改进代码

编写代码时，应运行并测试该代码以了解 bug 和性能。 使用 Visual Studio 的调试系统，可以调试在本地项目、远程设备或设备仿真器上运行的代码。 单步执行代码，一次执行一条语句，逐步检查变量。 或设置仅当指定条件为 true 时才命中的断点。 可以在代码编辑器中管理调试选项，因此无需离开代码。

有关 Visual Studio 中调试的详细信息，请参阅调试器入门。

若要提高应用性能，请查看 Visual Studio 分析功能。

Visual Studio 提供单元测试、Live Unit Testing、IntelliTest、负载和性能测试等测试选项。 Visual Studio 还拥有高级的代码分析功能，可查找设计、安全性和其他缺陷。

部署完成的应用程序

Visual Studio 具有可用于通过 Microsoft Store、SharePoint 站点或者 InstallShield 或 Windows Installer 技术将应用部署到用户或客户的工具。 可以通过 Visual Studio IDE 访问所有这些选项。 有关详细信息，请参阅部署应用程序、服务和组件。

管理源代码并与他人协作

在 Visual Studio 中，可以在任意提供商（包括 GitHub）托管的 Git 存储库中管理源代码。 还可以通过浏览来查找要连接到的 Azure DevOps Server。

如需完整的详细信息，请参阅 Visual Studio 中的 Git 体验页和 Visual Studio 版本控制文档导航页。 另外，有关如何使用 Visual Studio 连接到 Git 或 Azure DevOps 存储库的分步教程，请参阅从存储库打开项目页。

> **提示**
>
> 我们会继续构建 Git 功能集，并根据你的反馈对它进行迭代。 若要获取有关最新功能更新的详细信息以及可在其中共享反馈的调查的链接，请参阅 **Visual Studio 中的多存储库支持** 博客文章。

优化 Microsoft Dev Box 上的 Visual Studio 体验

> **备注**
>
> 此功能目前处于公开预览状态。 此信息与可能在发布之前进行了重大修改的功能相关。 Microsoft 不对此处提供的信息作任何明示或默示的担保。

使用 Visual Studio 17.7 预览版 3，可以预生成 Visual Studio 缓存并将其包含在开发箱映像中。 因此，Visual Studio 将会加载解决方案并在开发箱上更快地启用关键 IDE 功能。 还可以通过在开发箱映像中启用 Git commit-graph 优化来提高大型存储库的 Git 性能。

后续步骤

- 如果 Visual Studio 中没有你需要的确切功能，可以进行添加。 根据你的工作流和风格自定义 IDE，对尚未与 Visual Studio 集成的外部工具添加支持并修改现有的功能，以提高工作效率。 对于 Visual Studio 扩展性工具 (VS SDK) 的最新版本，请参阅 Visual Studio SDK。
- 可使用 .NET Compiler Platform（“Roslyn”）自行编写代码分析器和代码生成器。 在 Roslyn 中查找所需的一切内容。
- 查找由 Microsoft 开发人员以及 Visual Studio 开发社区创建的 Visual Studio 现有扩展。
- 若要了解有关扩展 Visual Studio 的详细信息，请参阅扩展 Visual Studio IDE。

另请参阅

- Visual Studio IDE 概述
- Visual Studio 2017 中的新增功能
- Visual Studio 2019 中的新增功能
- Visual Studio 2022 中的新增功能

教程：在 Visual Studio 中创建一个简单的 C# 控制台应用（第 1 部分，共 2 部分）

项目 • 2023/11/24

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本教程中，你将使用 Visual Studio 创建和运行 C# 控制台应用，并探索 Visual Studio 集成开发环境 (IDE) 的部分功能。 本教程是由两个部分构成的系列教程的第一部分。

在本教程中，请完成以下任务：

- ✓ 创建一个 Visual Studio 项目。
- ✓ 创建一个 C# 控制台应用。
- ✓ 调试应用。
- ✓ 关闭应用。
- ✓ 检查完整的代码。

在第 2 部分，你将扩展此应用以添加更多项目、了解调试技巧和引用第三方包。

必备条件

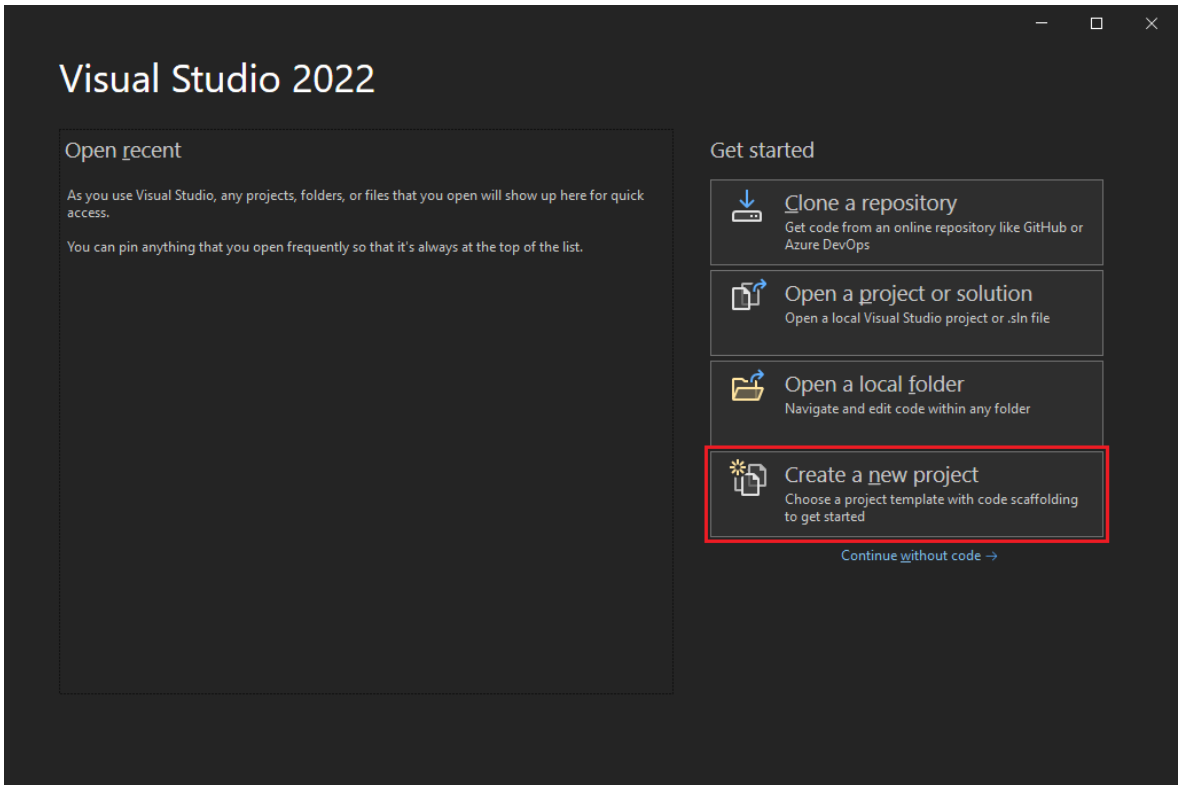
必须安装 Visual Studio。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

创建项目

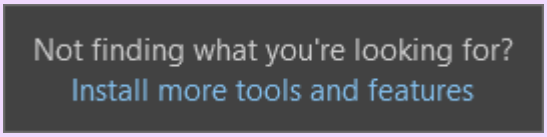
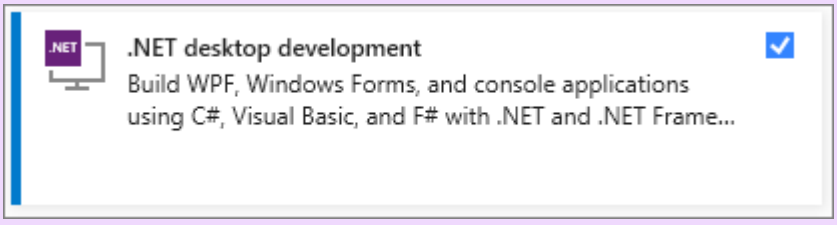
若要开始，请创建一个 C# 应用程序项目。 项目类型随附了所需的全部模板文件。

1. 打开 Visual Studio，然后在“开始”窗口中选择“创建新项目”。

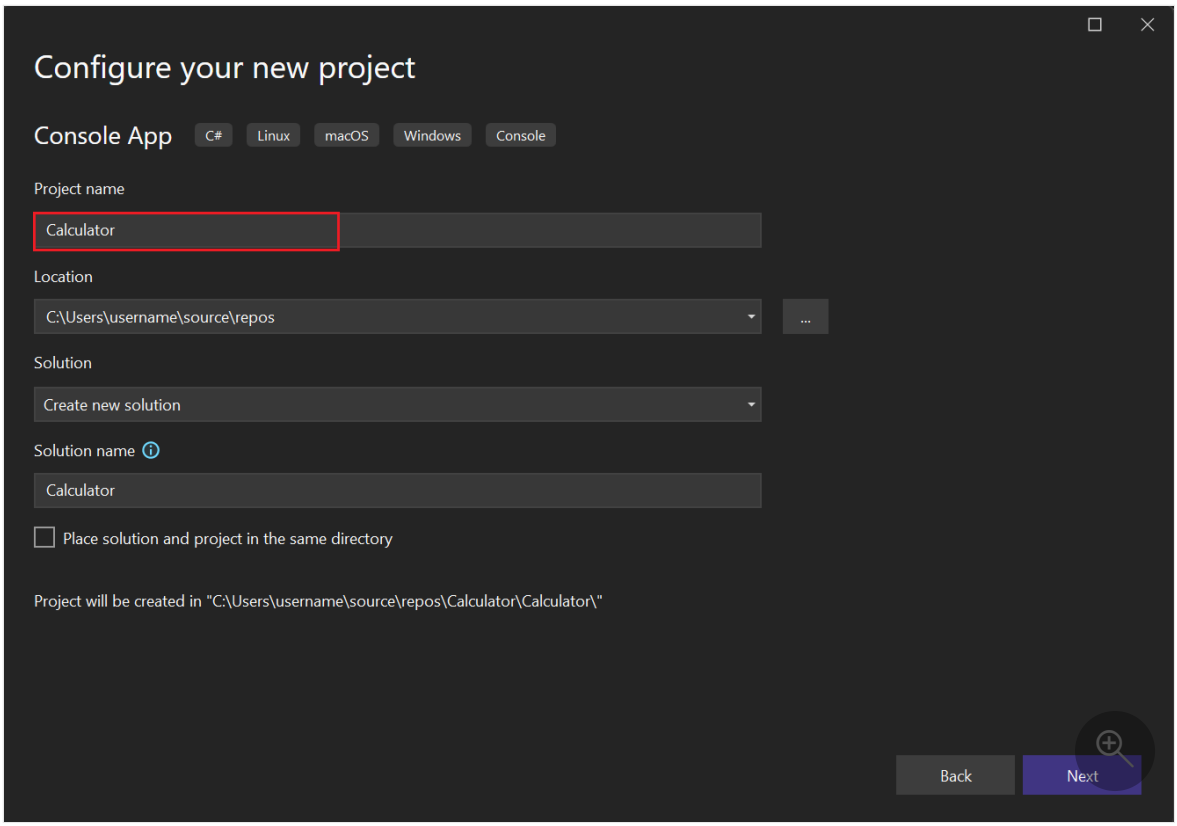


2. 在“创建新项目”窗口中选择“所有语言”，然后从下拉列表中选择“C#”。 从“所有平台”列表中选择“Windows”，然后从“所有项目类型”列表中选择“控制台”。

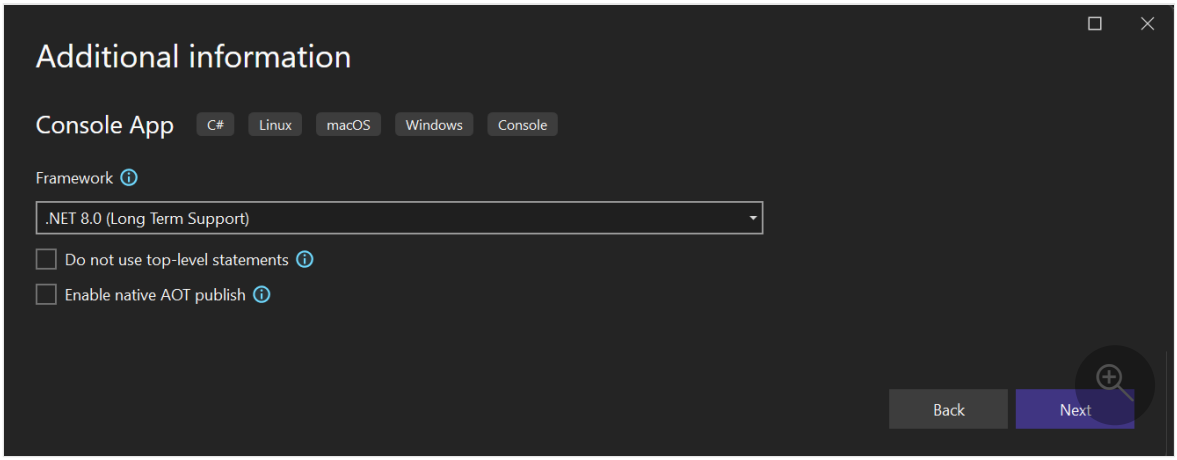
   应用语言、平台和项目类型筛选器后，选择“控制台应用”模板，然后选择“下一步”。

> **备注**
>
> 如果未看到“控制台应用”模板，请选择“安装更多工具和功能”。
>
> 
>
>
> 在 Visual Studio 安装程序中，选择“.NET 桌面开发”工作负载。
>
> 
>
>
> 在 Visual Studio 安装程序中，选择“修改”。 系统可能会提示你保存工作内容。 选择“继续”以安装该工作负载。
>
> 返回到“**创建项目**”过程中的步骤 2。

3. 在“配置新项目”窗口中，在“项目名称”框中键入“Calculator”，然后选择“下一步” 。



4. 在“其他信息”窗口中，为“目标框架”字段选择“.NET 8.0”。 然后选择“创建”。



Visual Studio 随即打开新项目，其中包含默认的“Hello World”代码。 如果要在编辑器中查看它，可以在“解决方案资源管理器”窗口中选择代码文件“Program.cs”，该窗口通常位于 Visual Studio 的右侧。

单个代码语句调用 WriteLine 方法在控制台窗口中显示文本字符串“Hello, World!”。 如果按 F5，则可以在调试模式下运行默认程序。 在调试器中运行应用程序后，控制台窗口将保持打开状态。 按任意键关闭控制台窗口。

> **备注**

从 .NET 6 开始，使用控制台模板的新项目会生成与以前版本不同的代码。 若要了解详细信息，请参阅**新的 C# 模板生成顶级语句**页。

创建应用

在本部分中，你将完成以下任务：

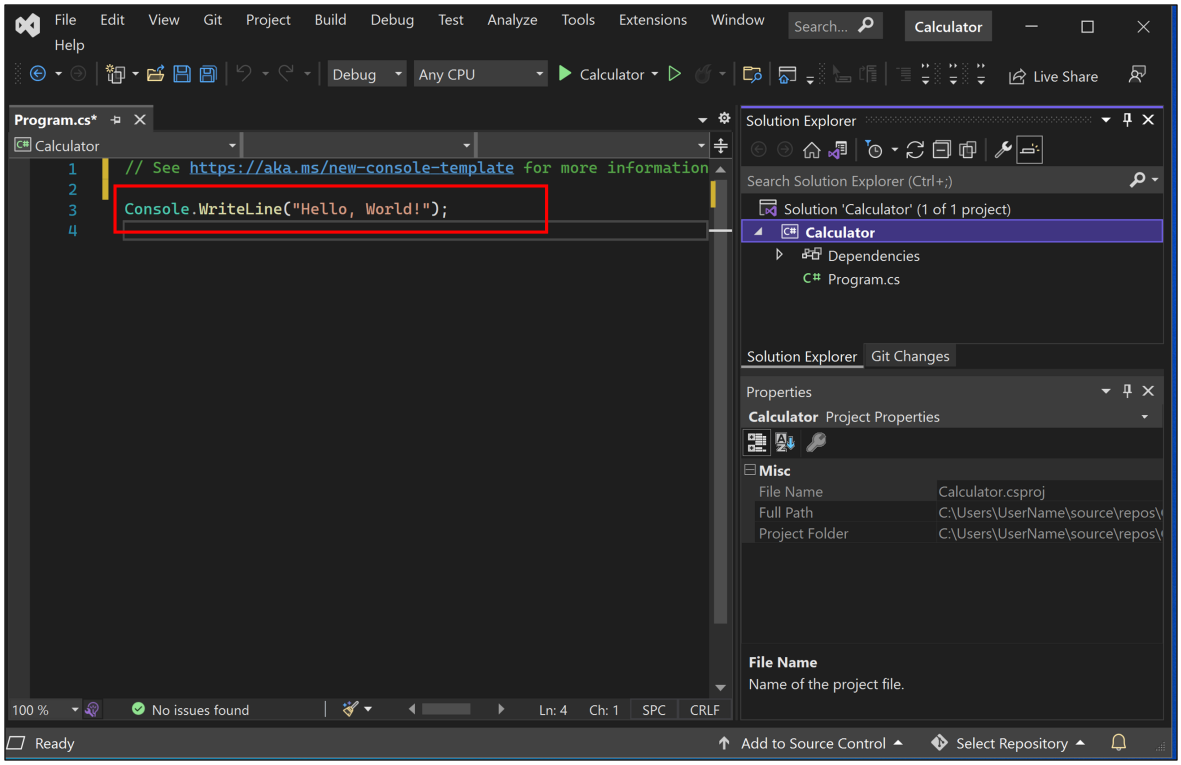
- 探索 C# 中的一些基本整数数学运算。
- 添加代码以创建一个基本计算器应用。
- 调试该应用，以查找并修复错误。
- 优化代码，使其更高效。

探索整数数学运算

首先在 C# 中进行一些基本的整数数学运算。

1. 在“解决方案资源管理器”的右侧窗格中选择“Program.cs”，以在代码编辑器中显示该文件

2. 在代码编辑器中，替换显示了 `Console.WriteLine("Hello World!");` 的默认“Hello World”代码。

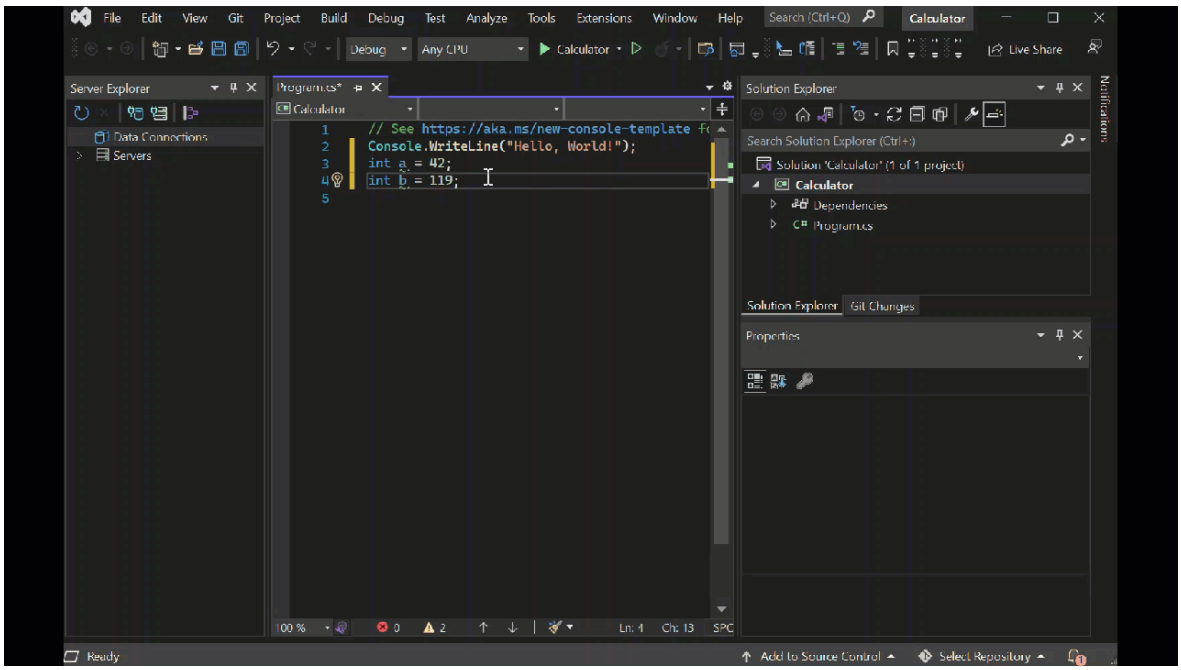


将该行替换为以下代码：

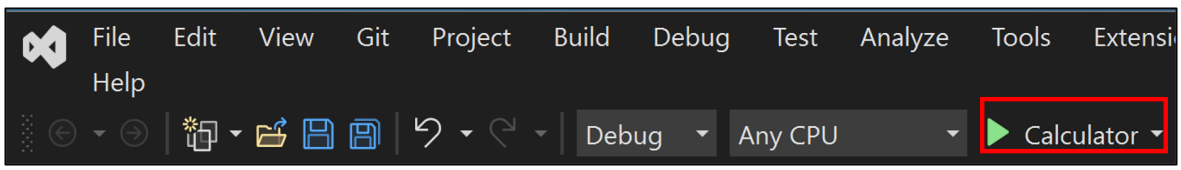
C#

```
int a = 42;
int b = 119;
int c = a + b;
Console.WriteLine(c);
Console.ReadKey();
```

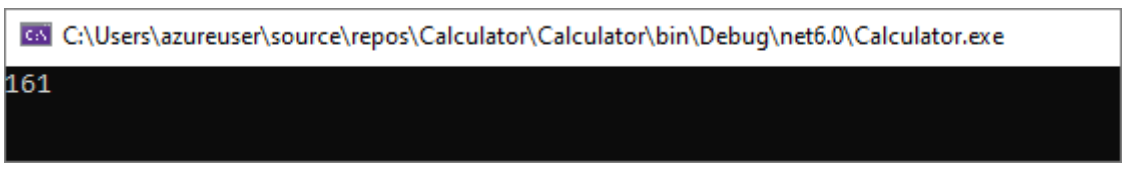
如果你输入代码，Visual Studio IntelliSense 功能将提供自动完成该条目的选项。



3. 若要生成并运行应用，请按 F5，或者在顶部工具栏中选择名称“Calculator”旁边的绿色箭头 。



随即会打开控制台窗口，其中显示了 42 + 119 的总和，即 161。



4. 关闭控制台窗口。

5. （可选）可以更改运算符来更改结果。 例如，可以将 `int c = a + b;` 代码行中的 `+` 运算符更改为 `-` 进行减法运算，更改为 `*` 进行乘法运算，或更改为 `/` 进行除法运算。 运行应用时，结果会相应地更改。

添加代码以创建计算器

向项目添加一组更复杂的计算器代码以继续操作。

1. 在代码编辑器中，将 Program.cs 中的所有代码替换为以下新代码：

C#

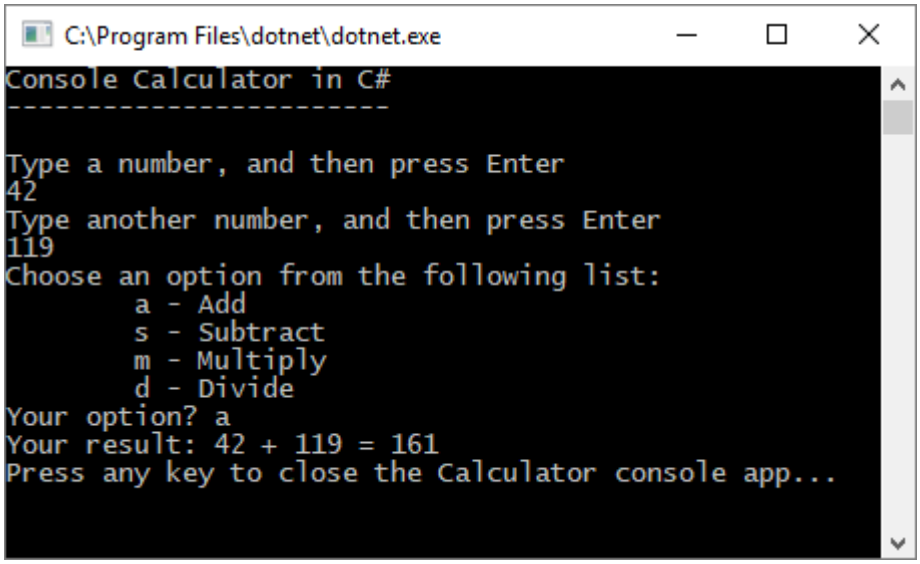
```
    // Declare variables and then initialize to zero.
    int num1 = 0; int num2 = 0;

    // Display title as the C# console calculator app.
    Console.WriteLine("Console Calculator in C#\r");
    Console.WriteLine("------------------------\n");

    // Ask the user to type the first number.
    Console.WriteLine("Type a number, and then press Enter");
    num1 = Convert.ToInt32(Console.ReadLine());

    // Ask the user to type the second number.
    Console.WriteLine("Type another number, and then press Enter");
    num2 = Convert.ToInt32(Console.ReadLine());

    // Ask the user to choose an option.
    Console.WriteLine("Choose an option from the following list:");
    Console.WriteLine("\ta - Add");
    Console.WriteLine("\ts - Subtract");
    Console.WriteLine("\tm - Multiply");
    Console.WriteLine("\td - Divide");
    Console.Write("Your option? ");

    // Use a switch statement to do the math.
    switch (Console.ReadLine())
    {
        case "a":
            Console.WriteLine($"Your result: {num1} + {num2} = " +
(num1 + num2));
            break;
        case "s":
            Console.WriteLine($"Your result: {num1} - {num2} = " +
(num1 - num2));
            break;
        case "m":
            Console.WriteLine($"Your result: {num1} * {num2} = " +
(num1 * num2));
            break;
        case "d":
            Console.WriteLine($"Your result: {num1} / {num2} = " +
(num1 / num2));
            break;
    }
    // Wait for the user to respond before closing.
    Console.Write("Press any key to close the Calculator console
app...");
    Console.ReadKey();
```

2. 选择“Calculator”按钮或按 F5 运行应用 。

   控制台窗口即会打开。

3. 在控制台窗口中，按照提示将数字 42 和 119 相加 。

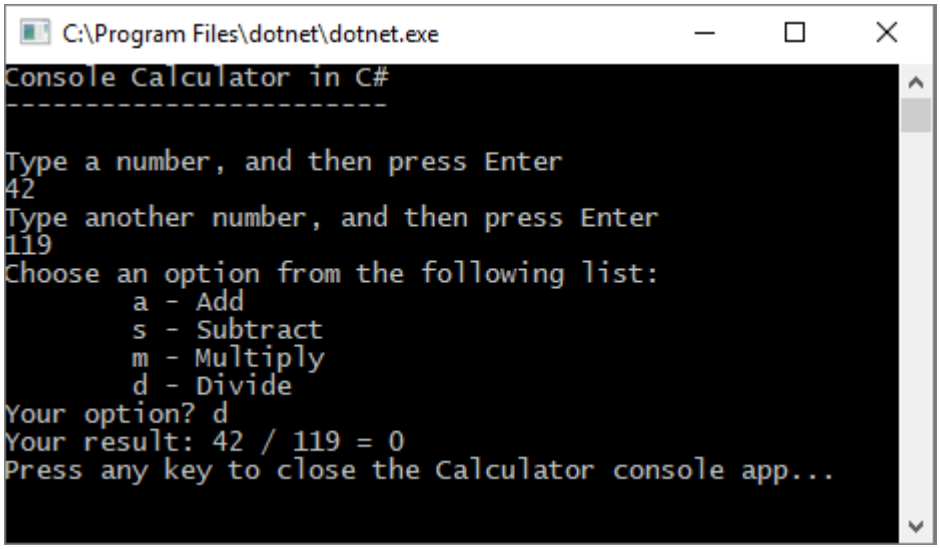
   应用应如以下屏幕快照所示：



添加小数运算功能

现在调整代码以添加更多功能。

当前的计算器应用仅接受并返回整数。 例如，如果运行该应用，将数字 42 除以数字 119，则结果为 0，这并不精确。



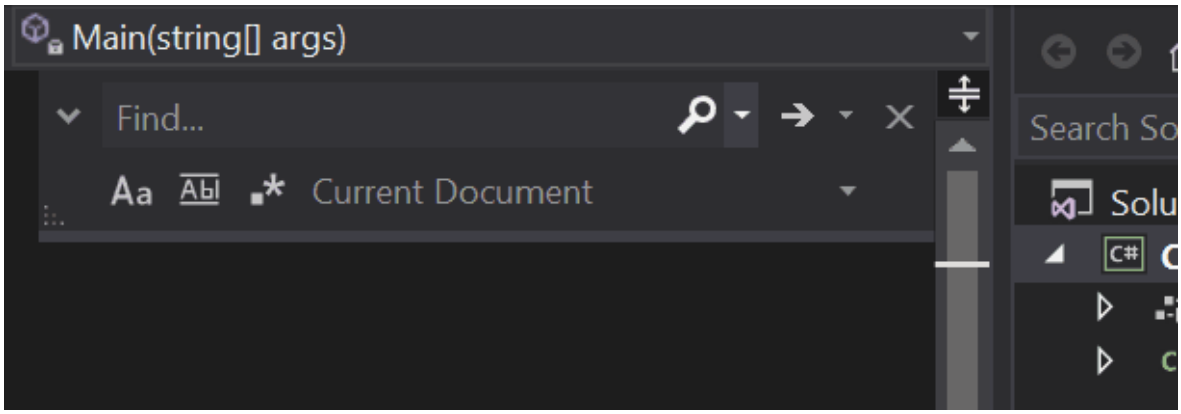
若要修复代码以通过处理小数来提高精度，请执行以下操作：

1. 在 Visual Studio 编辑器中的 Program.cs 内，按 Ctrl+H 打开“查找和替换”控件。

2. 在该控件中键入“int”，在“替换”字段中键入“float” 。

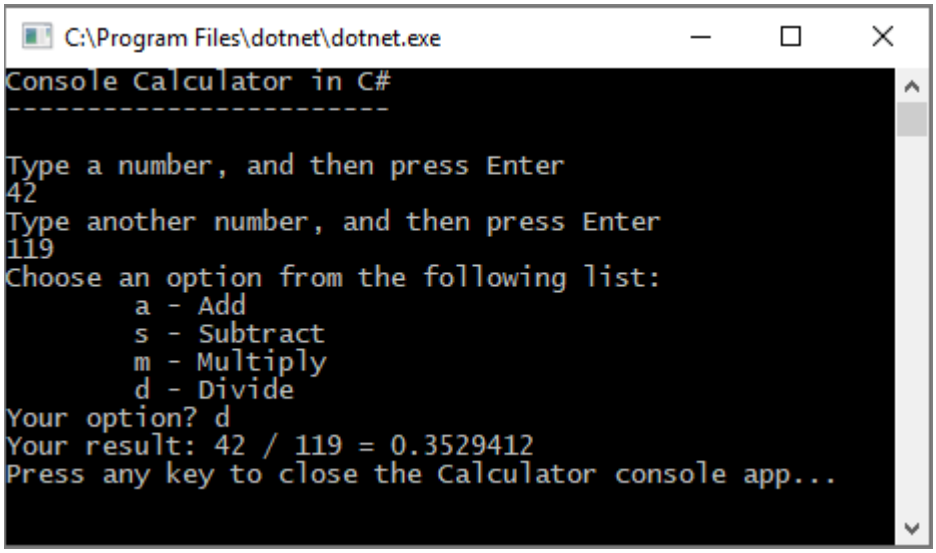
3. 在该控件中选择“匹配大小写”和“全字匹配”对应的图标，或者按 Alt+C 和 Alt+W 。

4. 选择“全部替换”图标或按 Alt+A 运行搜索和替换 。



5. 再次运行计算器应用，将数字 42 除以数字 119 。

   该应用现在返回的是小数而不是 0。

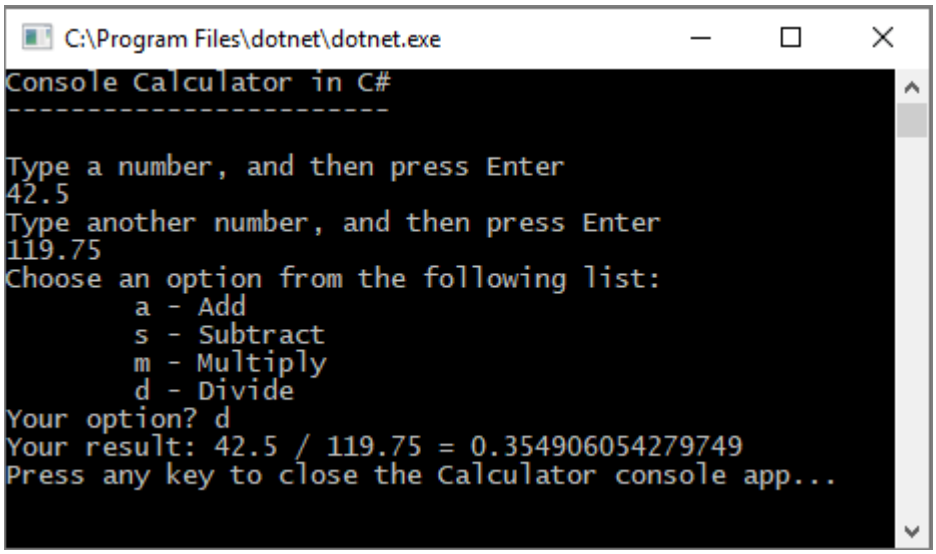


   现在，应用可以生成小数结果。 对代码再做一些调整，使应用也可以计算小数。

6. 使用“查找和替换”控件将 `float` 变量的每个实例更改为 `double`，并将 `Convert.ToInt32` 方法的每个实例更改为 `Convert.ToDouble`。

7. 运行计算器应用，将数字 42.5 除以数字 119.75 。

   该应用现在接受小数值，并返回更长的小数作为结果。



在修改代码部分，你将减少结果中的小数位数。

调试应用

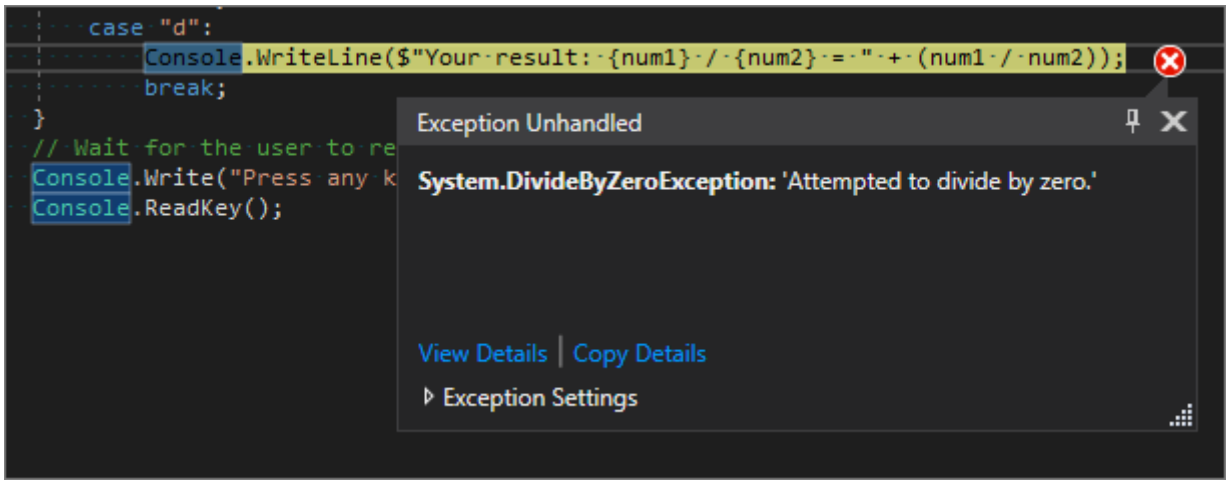
你改进了基本计算器应用，但该应用目前还不能处理异常，例如用户输入错误。 例如，如果用户尝试除以零或者输入意外的字符，则应用可能会停止工作、返回错误或返回意外的非数值结果。

现在来演练一些常见的用户输入错误，在调试程序中找到它们（若其出现），并在代码中修复它们。

> **提示**
>
> 有关调试器及其工作原理的详细信息，请参阅**初步了解 Visual Studio 调试器**。

修复“被零除”错误

如果你尝试将数字除以零，则控制台应用可能会冻结，并在代码编辑器中显示哪些内容是错误的。



> **备注**
>
> 有时，应用不会冻结且调试器不会显示“被零除”错误。 相反，应用可能会返回意外的非数字结果，如无穷符号。 以下代码修复仍然适用。

现在更改代码以解决此错误。 在 Program.cs 中，将 `case "d":` 的代码替换为以下代码：

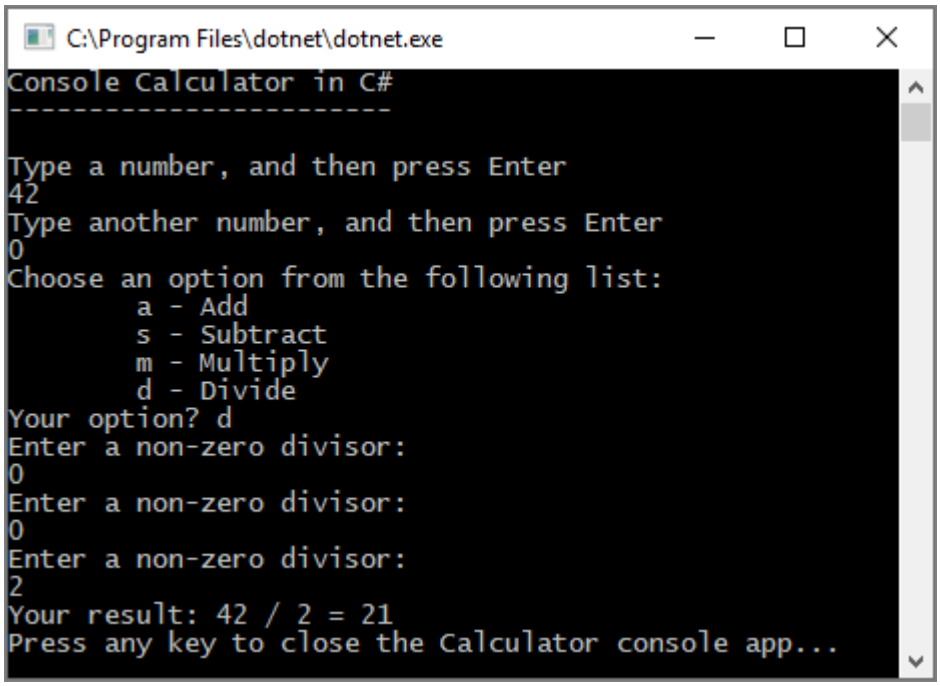
C#

```
            // Ask the user to enter a non-zero divisor until they do so.
                while (num2 == 0)
                {
                    Console.WriteLine("Enter a non-zero divisor: ");
                    num2 = Convert.ToInt32(Console.ReadLine());
                }
                Console.WriteLine($"Your result: {num1} / {num2} = " + (num1
/ num2));
                break;
        }
```

替换代码后，包含 `switch` 语句的部分应如以下屏幕截图所示：

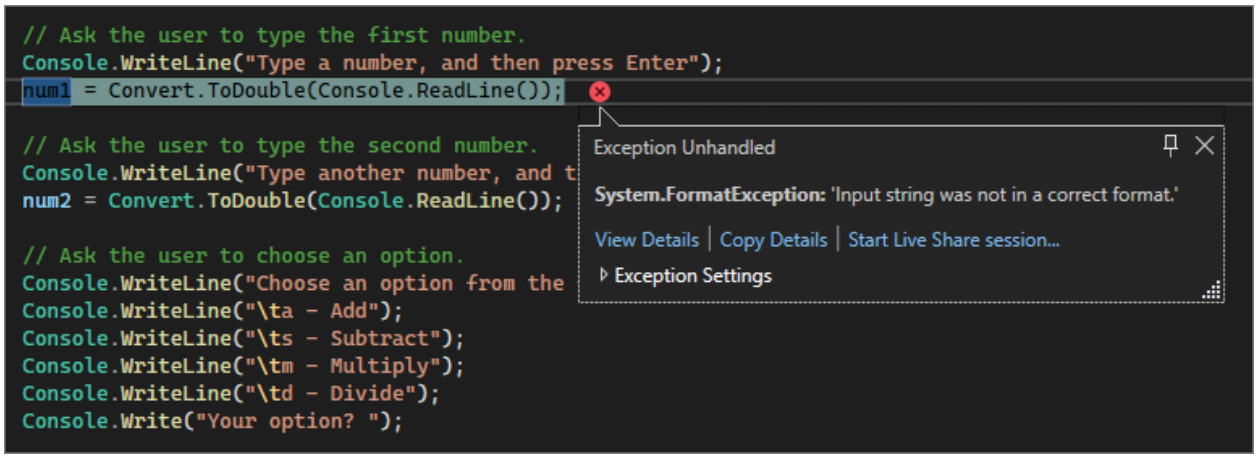
```
// Use a switch statement to do the math
switch (Console.ReadLine())
{
    case "a":
        Console.WriteLine($"Your result: {num1} + {num2} = " + (num1 + num2));
        break;
    case "s":
        Console.WriteLine($"Your result: {num1} - {num2} = " + (num1 - num2));
        break;
    case "m":
        Console.WriteLine($"Your result: {num1} * {num2} = " + (num1 * num2));
        break;
    case "d":
        // Ask the user to enter a non-zero divisor until they do so
        while (num2 == 0)
        {
            Console.WriteLine("Enter a non-zero divisor: ");
            num2 = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine($"Your result: {num1} / {num2} = " + (num1 / num2));
        break;
}
// Wait for the user to respond before closing
Console.Write("Press any key to close the Calculator console app...");
Console.ReadKey();
```

现在，将任意数字除以零时，该应用会要求输入另一个数字，并且在你提供非零数字之前，它会不断地要求你这样做。



修复“格式”错误

如果在应用要求输入数字字符时你输入了字母字符，应用将会冻结。 Visual Studio 将显示代码编辑器中的哪些内容是错误的。



若要防止此异常，可以重构先前输入的代码。

修改代码

可将应用分为两个类：`Calculator` 和 `Program`，而不是依赖于 `program` 类来处理所有代码。

`Calculator` 类处理大量的计算工作，而 `Program` 类则会处理用户界面和错误处理工作。

现在就开始吧。

1. 在 Program.cs 中，删除所有内容并添加以下新的 `Calculator` 类：

C#

```
class Calculator
{
    public static double DoOperation(double num1, double num2, string
op)
    {
        double result = double.NaN; // Default value is "not-a-number"
if an operation, such as division, could result in an error.

        // Use a switch statement to do the math.
        switch (op)
        {
            case "a":
                result = num1 + num2;
                break;
            case "s":
                result = num1 - num2;
                break;
            case "m":
                result = num1 * num2;
                break;
            case "d":
                // Ask the user to enter a non-zero divisor.
                if (num2 != 0)
```

```
                {
                    result = num1 / num2;
                }
                break;
            // Return text for an incorrect option entry.
            default:
                break;
        }
        return result;
    }
}
```

2. 另外还添加新的 `Program` 类，如下所示：

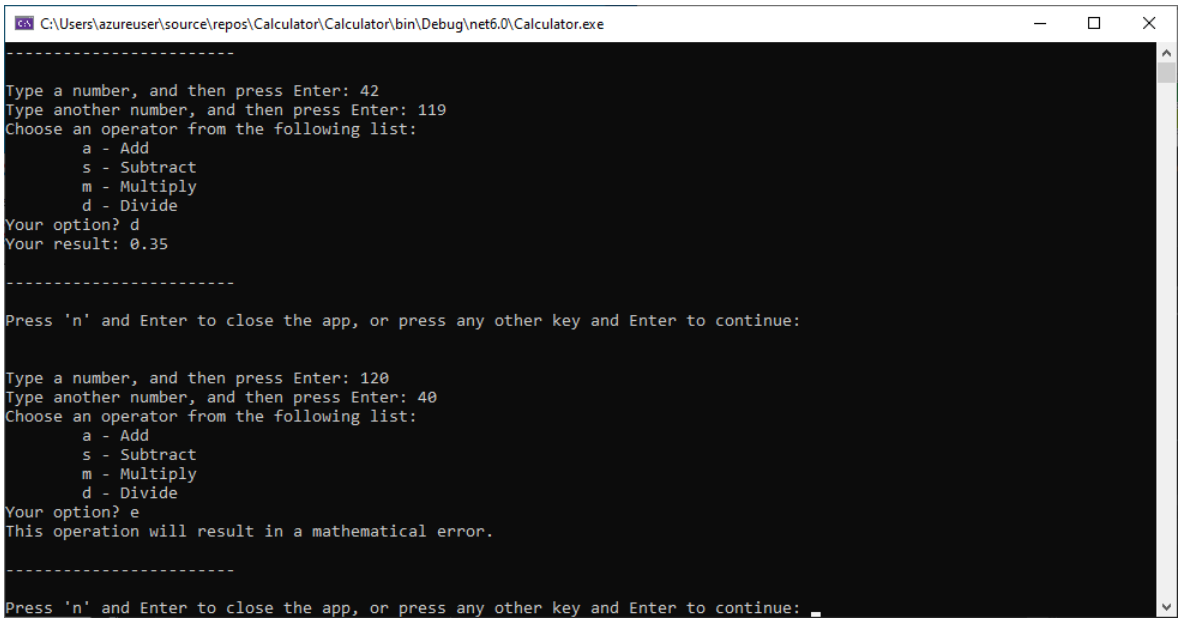
C#

```
class Program
{
    static void Main(string[] args)
    {
        bool endApp = false;
        // Display title as the C# console calculator app.
        Console.WriteLine("Console Calculator in C#\r");
        Console.WriteLine("------------------------\n");

        while (!endApp)
        {
            // Declare variables and set to empty.
            string numInput1 = "";
            string numInput2 = "";
            double result = 0;

            // Ask the user to type the first number.
            Console.Write("Type a number, and then press Enter: ");
            numInput1 = Console.ReadLine();

            double cleanNum1 = 0;
            while (!double.TryParse(numInput1, out cleanNum1))
            {
                Console.Write("This is not valid input. Please enter an
integer value: ");
                numInput1 = Console.ReadLine();
            }

            // Ask the user to type the second number.
            Console.Write("Type another number, and then press Enter:
");
            numInput2 = Console.ReadLine();

            double cleanNum2 = 0;
            while (!double.TryParse(numInput2, out cleanNum2))
            {
```

```
                Console.Write("This is not valid input. Please enter an
integer value: ");
                numInput2 = Console.ReadLine();
            }

            // Ask the user to choose an operator.
            Console.WriteLine("Choose an operator from the following
list:");
            Console.WriteLine("\ta - Add");
            Console.WriteLine("\ts - Subtract");
            Console.WriteLine("\tm - Multiply");
            Console.WriteLine("\td - Divide");
            Console.Write("Your option? ");

            string op = Console.ReadLine();

            try
            {
                result = Calculator.DoOperation(cleanNum1, cleanNum2,
op);
                if (double.IsNaN(result))
                {
                    Console.WriteLine("This operation will result in a
mathematical error.\n");
                }
                else Console.WriteLine("Your result: {0:0.##}\n",
result);
            }
            catch (Exception e)
            {
                Console.WriteLine("Oh no! An exception occurred trying
to do the math.\n - Details: " + e.Message);
            }

            Console.WriteLine("-----------------------\n");

            // Wait for the user to respond before closing.
            Console.Write("Press 'n' and Enter to close the app, or
press any other key and Enter to continue: ");
            if (Console.ReadLine() == "n") endApp = true;

            Console.WriteLine("\n"); // Friendly linespacing.
        }
        return;
    }
}
```

3. 选择“Calculator”按钮或按 F5 运行应用 。

4. 按照提示，用数字 42 除以数字 119 。 结果应类似于以下屏幕截图：

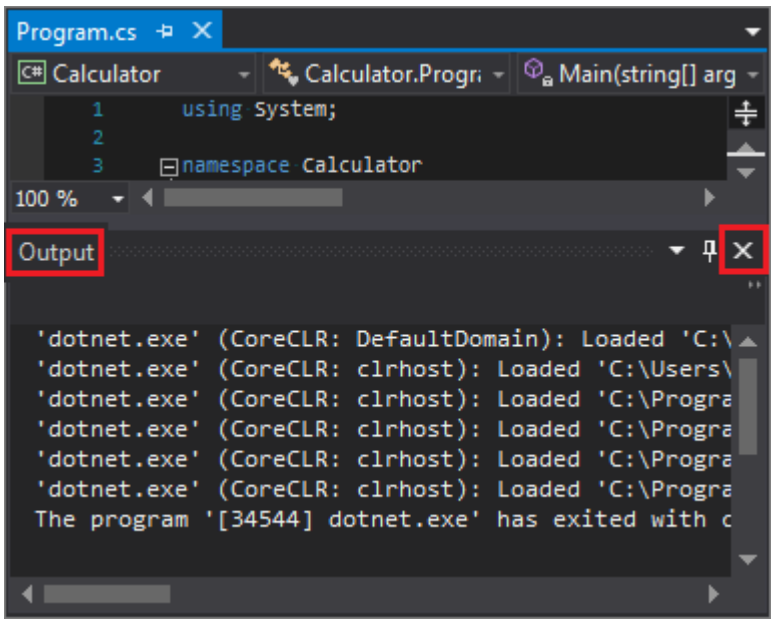


现在可以运行更多计算，直到选择关闭控制台应用。 结果中的小数位数也会减少。 如果输入错误的字符，则会得到相应的错误响应。

关闭应用程序

1. 如果尚未关闭计算器应用，现在请将其关闭。

2. 关闭 Visual Studio 中的输出窗格。



3. 在 Visual Studio 中，按 Ctrl+S 保存应用 。

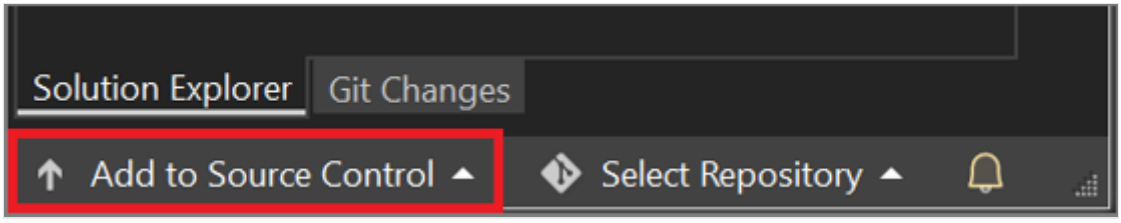
添加 Git 源代码管理

现在你已经创建了应用，可能需要将它添加到 Git 存储库。 Visual Studio 通过 Git 工具简化了该过程，你可直接从 IDE 中使用这些工具。

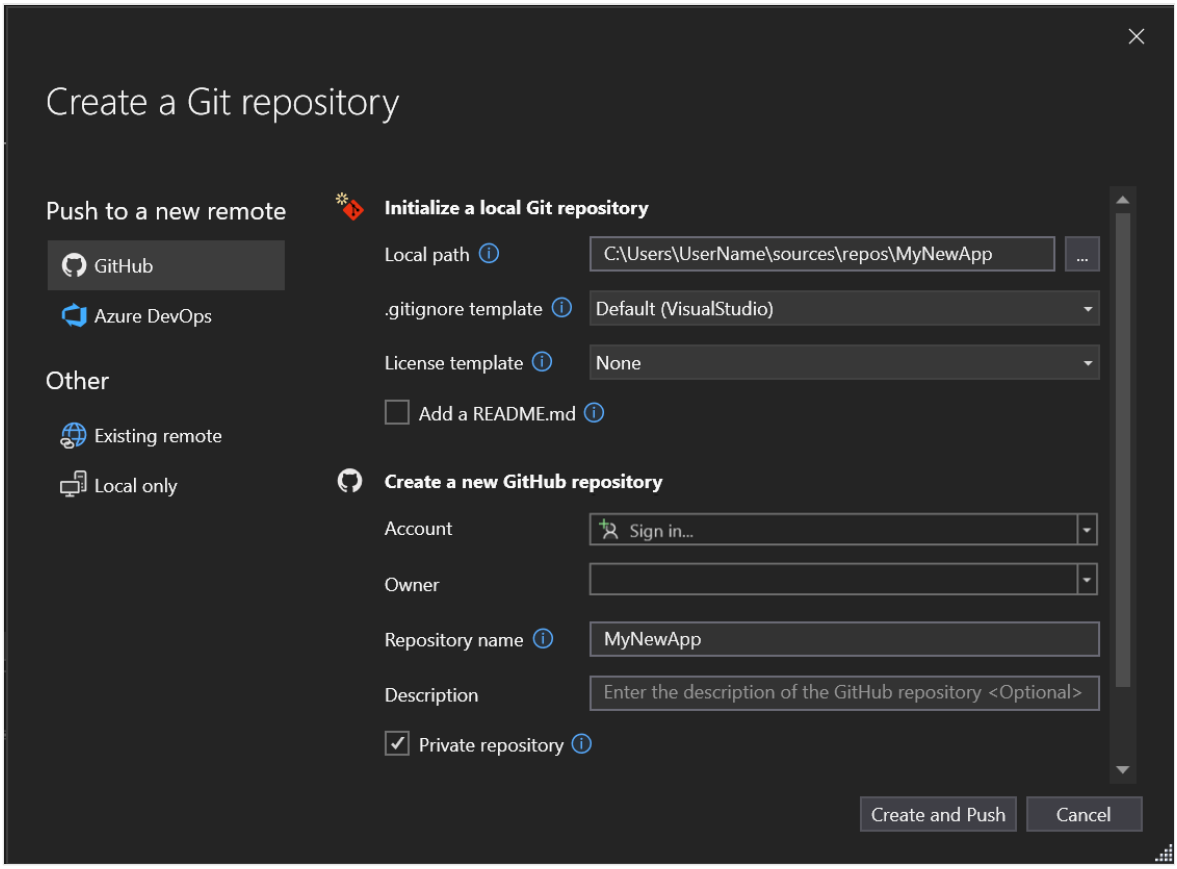
> **提示**
>
> Git 是使用最广泛的新式版本控制系统，因此无论你是专业开发人员，还是正在学习如何编码，Git 都非常有用。 如果你是刚刚接触 Git，可访问 **https://git-scm.com/** 网站开始了解。 在这里，你能找到速查表、畅销在线图书和 Git 基础知识视频。

若要将代码与 Git 关联，需要首先创建一个新的 Git 存储库来容纳代码：

1. 在 Visual Studio 右下角的状态栏中，选择“添加到源代码管理”，然后选择“Git” 。



2. 在“创建 Git 存储库”对话框中，登录到 GitHub。



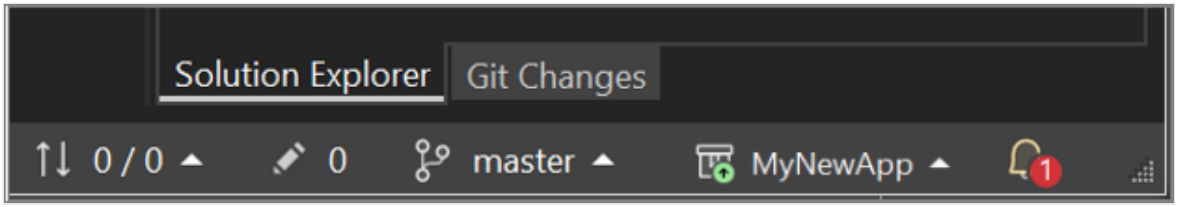
存储库名称根据你的文件夹位置自动填充。 默认情况下，新存储库是专用的，这意味着只有你可以访问它。

> **提示**

> 无论存储库是公用的还是专用的，都最好将代码的远程备份安全地存储在 GitHub 上。 即使你不与团队合作，也可使用任意计算机上在远程存储库中访问你的代码。

3. 选择“创建并推送”。

   创建存储库后，状态栏中会显示状态详细信息。

   带箭头的第一个图标显示当前分支中的传出/传入提交数。 可以使用此图标来拉取任何传入提交或推送任何传出提交。 还可选择先查看这些提交。 为此，请选择图标，然后选择“查看传出/传入”。

   带铅笔的第二个图标显示代码的未提交更改数。 可选择此图标，在“Git 更改”窗口中查看这些更改。

若要详细了解如何在应用中使用 Git，请参阅 Visual Studio 版本控制文档。

查看：代码完成

在本教程中，你对计算器应用做出了许多更改。 该应用现在可以更高效地处理计算资源，并可处理大多数的用户输入错误。

以下是完整代码，全部显示在同一个位置：

C#

```
class Calculator
{
    public static double DoOperation(double num1, double num2, string op)
    {
        double result = double.NaN; // Default value is "not-a-number" which
we use if an operation, such as division, could result in an error.

        // Use a switch statement to do the math.
        switch (op)
        {
            case "a":
                result = num1 + num2;
                break;
            case "s":
                result = num1 - num2;
```

```
                break;
            case "m":
                result = num1 * num2;
                break;
            case "d":
                // Ask the user to enter a non-zero divisor.
                if (num2 != 0)
                {
                    result = num1 / num2;
                }
                break;
            // Return text for an incorrect option entry.
            default:
                break;
        }
        return result;
    }
}

class Program
{
    static void Main(string[] args)
    {
        bool endApp = false;
        // Display title as the C# console calculator app.
        Console.WriteLine("Console Calculator in C#\r");
        Console.WriteLine("------------------------\n");

        while (!endApp)
        {
            // Declare variables and set to empty.
            string numInput1 = "";
            string numInput2 = "";
            double result = 0;

            // Ask the user to type the first number.
            Console.Write("Type a number, and then press Enter: ");
            numInput1 = Console.ReadLine();

            double cleanNum1 = 0;
            while (!double.TryParse(numInput1, out cleanNum1))
            {
                Console.Write("This is not valid input. Please enter an
integer value: ");
                numInput1 = Console.ReadLine();
            }

            // Ask the user to type the second number.
            Console.Write("Type another number, and then press Enter: ");
            numInput2 = Console.ReadLine();

            double cleanNum2 = 0;
            while (!double.TryParse(numInput2, out cleanNum2))
            {
                Console.Write("This is not valid input. Please enter an
```

```
integer value: ");
                numInput2 = Console.ReadLine();
            }

            // Ask the user to choose an operator.
            Console.WriteLine("Choose an operator from the following
list:");
            Console.WriteLine("\ta - Add");
            Console.WriteLine("\ts - Subtract");
            Console.WriteLine("\tm - Multiply");
            Console.WriteLine("\td - Divide");
            Console.Write("Your option? ");

            string op = Console.ReadLine();

            try
            {
                result = Calculator.DoOperation(cleanNum1, cleanNum2, op);
                if (double.IsNaN(result))
                {
                    Console.WriteLine("This operation will result in a
mathematical error.\n");
                }
                else Console.WriteLine("Your result: {0:0.##}\n", result);
            }
            catch (Exception e)
            {
                Console.WriteLine("Oh no! An exception occurred trying to do
the math.\n - Details: " + e.Message);
            }

            Console.WriteLine("------------------------\n");

            // Wait for the user to respond before closing.
            Console.Write("Press 'n' and Enter to close the app, or press
any other key and Enter to continue: ");
            if (Console.ReadLine() == "n") endApp = true;

            Console.WriteLine("\n"); // Friendly linespacing.
        }
        return;
    }
}
```

后续步骤

继续学习本教程的第二部分：

教程第 2 部分：扩展和调试 C# 控制台应用

教程：扩展 C# 控制台应用并在 Visual Studio 中调试（第 2 部分，共 2 部分）

项目 • 2023/10/31

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本教程系列的第 2 部分，你将更深入地了解日常开发所需的 Visual Studio 生成和调试功能。 这些功能包括管理多个项目、调试和引用第三方包。 你将运行在本教程的第 1 部分中创建的 C# 控制台应用，并在此过程中了解 Visual Studio 集成开发环境 (IDE) 的部分功能。 本教程是由两个部分构成的系列教程的第二部分。

在本教程中，请完成以下任务：

- ✓ 添加第二个项目。
- ✓ 参考库并添加包。
- ✓ 调试代码。
- ✓ 检查完整的代码。

先决条件

按照本文进行操作时，可以使用以下任一计算器应用：

- 本教程系列第 1 部分中的计算器控制台应用。
- vs-tutorial-samples 存储库 中的 C# 计算器应用。 要开始，请从存储库中打开应用。

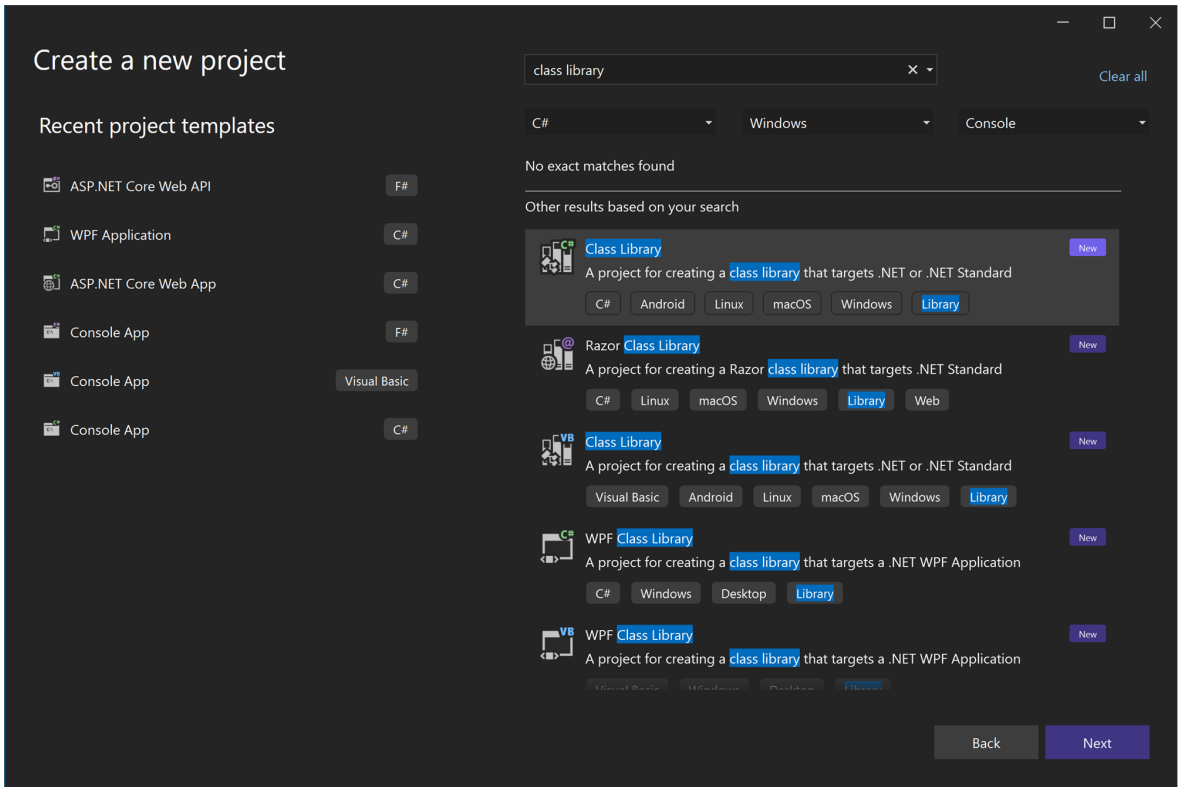
再添加一个项目

实际代码涉及在一个解决方案中协同工作的多个项目。 你可以将类库项目添加到计算器应用，该应用提供一些计算器功能。

在 Visual Studio，可以使用菜单命令“文件”>“添加”>“新项目”来添加新项目 。 也可以右键单击“解决方案资源管理器”中的解决方案，从上下文菜单中添加项目。

1. 在“解决方案资源管理器”中，右键单击解决方案节点，然后选择“添加”>“新建项目”。

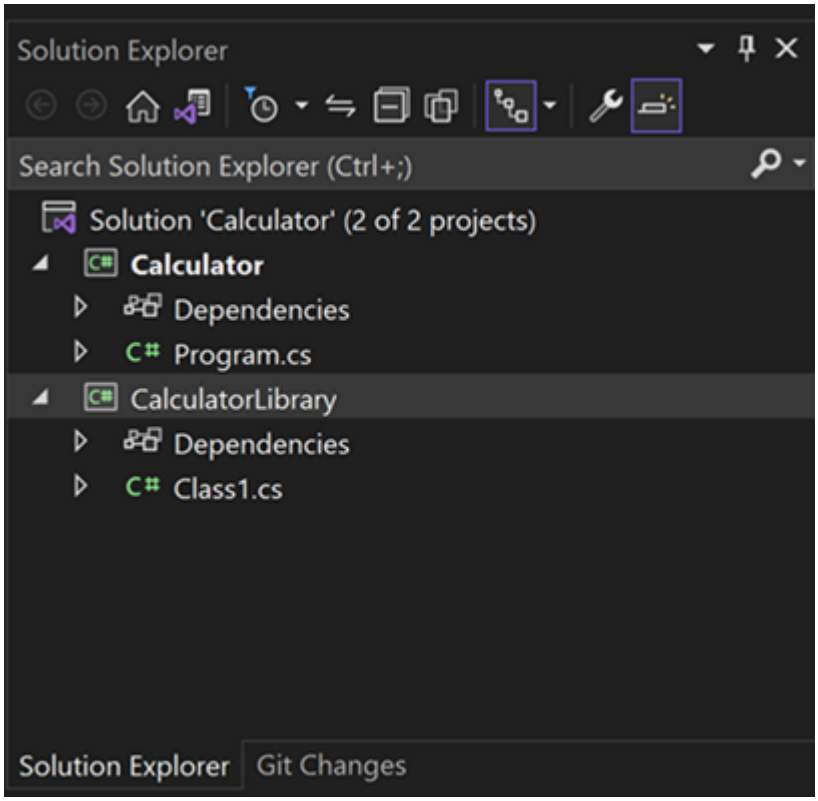
2. 在“添加新项目”窗口的搜索框中键入“类库”。 选择 C#“类库”项目模板，然后选择“下一步” 。



3. 在“配置新项目”屏幕中，键入项目名称 CalculatorLibrary，然后选择“下一步”。

4. 在“其他信息”屏幕上，“.NET 8.0”被选中。 选择**创建**。

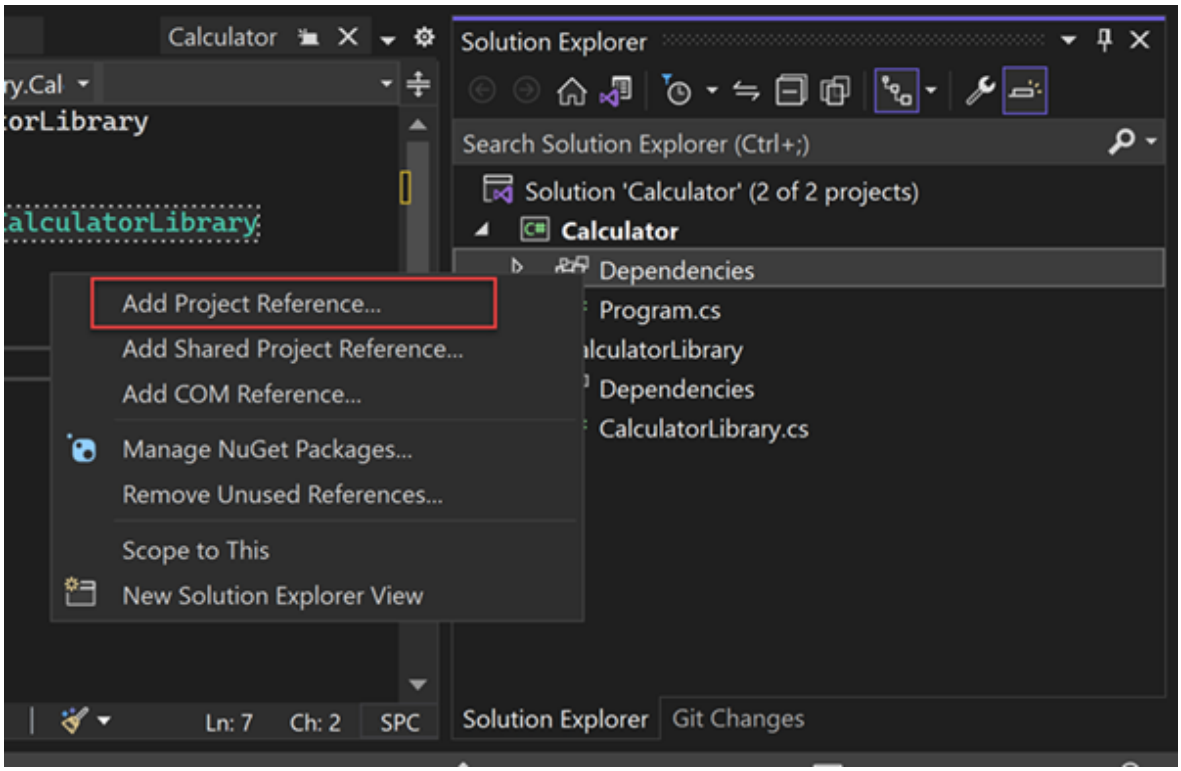
   Visual Studio 将创建新项目并将其添加到解决方案中。



5. 将 Class1.cs 文件重命名为 CalculatorLibrary.cs 。 要重命名文件，可以在“解决方案资源管理器”中右键单击该名称，然后选择“重命名”，选择该名称并按 F2，或选择该名称，然后再次选择以键入。

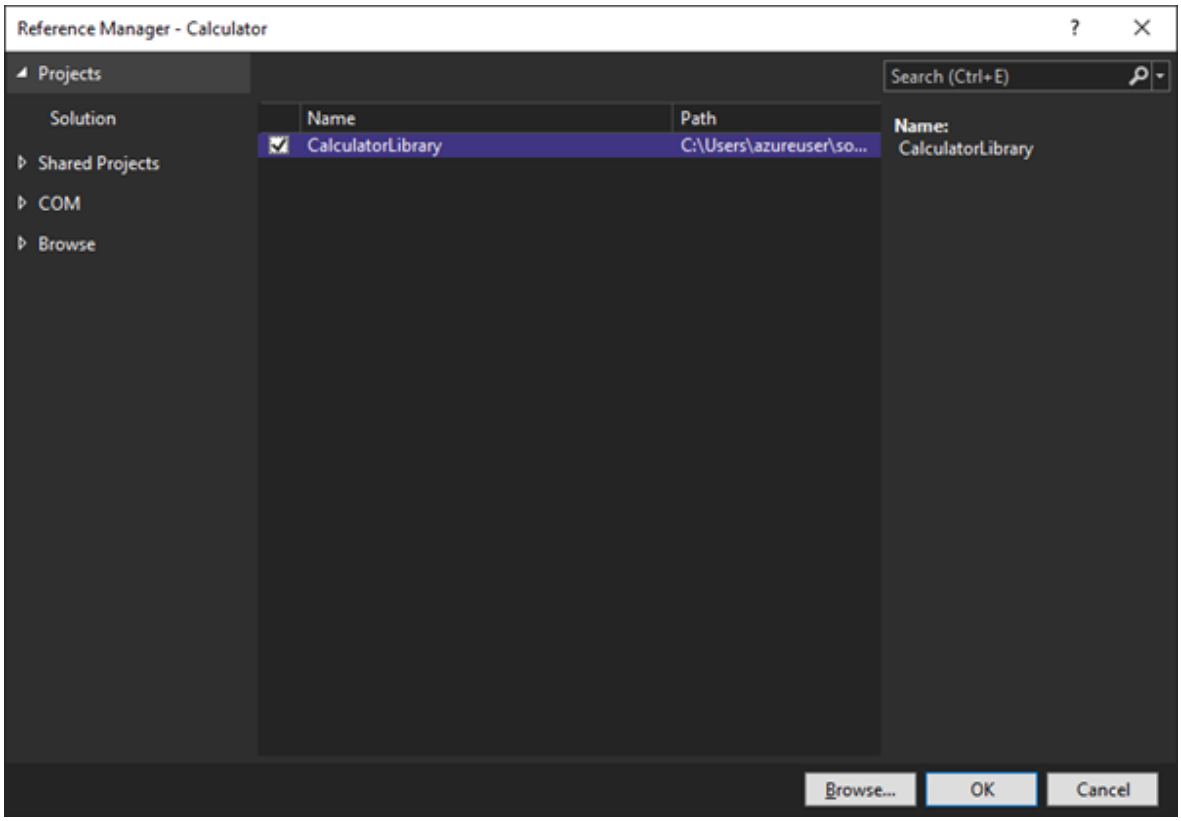
此时可能会出现一条消息，询问你是否要重命名文件中对 `Class1` 的所有引用。 你可以随意回答，因为你将在后续步骤中替换该代码。

6. 现在添加项目引用，以便第一个项目可以使用新类库公开的 API。 右键单击 Calculator 项目中的“依赖项”节点，然后选择“添加项目引用” 。

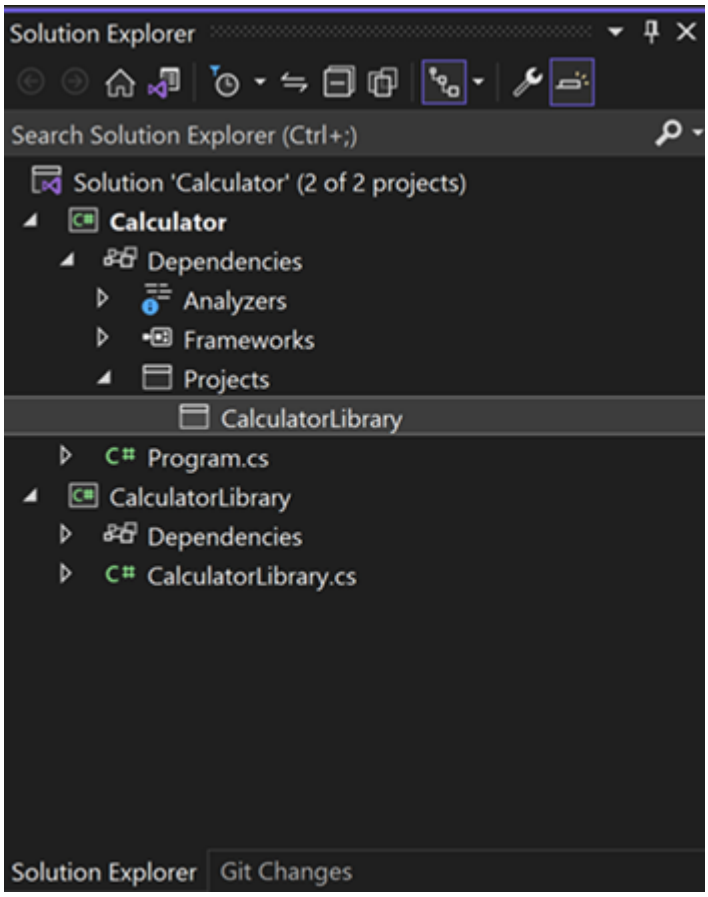


此时将显示“引用管理器”对话框。 在此对话框中，可以添加对项目所需的其他项目、程序集和 COM DLL 的引用。

7. 在“引用管理器”对话框中，选中 CalculatorLibrary 项目对应的复选框，然后选择“确定” 。



项目引用显示在解决方案资源管理器中的“项目”节点下 。



8. 在 Program.cs 中，选择 `Calculator` 类及其所有代码，然后按 Ctrl+X 进行剪切 。然后，在 CalculatorLibrary.cs 中，将代码粘贴到 `CalculatorLibrary` 命名空间中。

   然后，在 Calculator 类之前添加 `public`，以在库外部公开它。

CalculatorLibrary.cs 现在应类似于以下代码：

C#

```
 namespace CalculatorLibrary
 {
     public class Calculator
     {
         public static double DoOperation(double num1, double num2,
string op)
         {
             double result = double.NaN; // Default value is "not-a-
number" if an operation, such as division, could result in an error.

             // Use a switch statement to do the math.
             switch (op)
             {
                 case "a":
                     result = num1 + num2;
                     break;
                 case "s":
                     result = num1 - num2;
                     break;
                 case "m":
                     result = num1 * num2;
                     break;
                 case "d":
                     // Ask the user to enter a non-zero divisor.
                     if (num2 != 0)
                     {
                         result = num1 / num2;
                     }
                     break;
                 // Return text for an incorrect option entry.
                 default:
                     break;
             }
             return result;
         }
     }
 }
```

9. Program.cs 也有引用，但有错误指出 `Calculator.DoOperation` 调用无法解决。 该错误是因为 `CalculatorLibrary` 位于不同的命名空间中。 对于完全限定的引用，可以将 `CalculatorLibrary` 命名空间添加到 `Calculator.DoOperation` 调用：

C#

```
result = CalculatorLibrary.Calculator.DoOperation(cleanNum1, cleanNum2,
op);
```

或者，你可以尝试将 `using` 指令添加到文件的开头：

```
C#

using CalculatorLibrary;
```

添加 `using` 指令应该可以让你从调用站点中删除 `CalculatorLibrary` 命名空间，但现在存在歧义。 `Calculator` 是 `CalculatorLibrary` 中的类？还是 `Calculator` 是命名空间？

为了解决歧义，请在 Program.cs 中将命名空间从 `Calculator` 重命名为 `CalculatorProgram`。

```
C#

namespace CalculatorProgram
```

引用 .NET 库：写入日志

你可以使用 .NET Trace 类添加所有操作日志，并将其写入文本文件。 `Trace` 类对于基本的打印调试技术也很有用。 `Trace` 类位于 `System.Diagnostics` 中，并使用 `System.IO` 类（如 `StreamWriter`）。

1. 首先，在 CalculatorLibrary.cs 的顶部添加 `using` 指令：

```
C#

using System.Diagnostics;
```

2. `Trace` 类的这种用法必须保留对该类的引用，它与文件流关联。 这意味着计算器作为一个对象的工作性能将提升，所以应在 CalculatorLibrary.cs 中的 `Calculator` 类的开头添加一个构造函数。

   此外，还应删除 `static` 关键字以将静态 `DoOperation` 方法更改为成员方法。

```
C#

public Calculator()
   {
        StreamWriter logFile = File.CreateText("calculator.log");
        Trace.Listeners.Add(new TextWriterTraceListener(logFile));
        Trace.AutoFlush = true;
        Trace.WriteLine("Starting Calculator Log");
```

```
        Trace.WriteLine(String.Format("Started {0}",
System.DateTime.Now.ToString()));
    }

public double DoOperation(double num1, double num2, string op)
    {
```

3. 将日志输出添加到每个计算。 `DoOperation` 现在应如以下代码所示：

C#

```
public double DoOperation(double num1, double num2, string op)
{
     double result = double.NaN; // Default value is "not-a-number" if
an operation, such as division, could result in an error.

     // Use a switch statement to do the math.
     switch (op)
     {
         case "a":
             result = num1 + num2;
             Trace.WriteLine(String.Format("{0} + {1} = {2}", num1,
num2, result));
             break;
         case "s":
             result = num1 - num2;
             Trace.WriteLine(String.Format("{0} - {1} = {2}", num1,
num2, result));
             break;
         case "m":
             result = num1 * num2;
             Trace.WriteLine(String.Format("{0} * {1} = {2}", num1,
num2, result));
             break;
         case "d":
             // Ask the user to enter a non-zero divisor.
             if (num2 != 0)
             {
                 result = num1 / num2;
                 Trace.WriteLine(String.Format("{0} / {1} = {2}", num1,
num2, result));
             }
                 break;
         // Return text for an incorrect option entry.
         default:
             break;
     }
     return result;
 }
```

4. 回到 Program.cs，红色波浪下划线现在标记静态调用。 要修复此问题，请通过紧邻 `while (!endApp)` 循环前方添加以下代码行来创建一个 `calculator` 变量：

   C#

   ```
   Calculator calculator = new Calculator();
   ```

   此外，还应修改 `DoOperation` 调用站点以引用以小写 `calculator` 命名的对象。 代码现在是成员调用，而不是对静态方法的调用。

   C#

   ```
   result = calculator.DoOperation(cleanNum1, cleanNum2, op);
   ```

5. 再次运行应用。 完成后，右键单击 Calculator 项目节点，然后选择“在文件资源管理器中打开文件夹”。

6. 在文件资源管理器中，导航到 bin/Debug/ 下的 output 文件夹，然后打开 calculator.log 文件 。 输出应类似于：

   输出

   ```
   Starting Calculator Log
   Started 7/9/2020 1:58:19 PM
   1 + 2 = 3
   3 * 3 = 9
   ```

此时，CalculatorLibrary.cs 应类似于以下代码：

C#

```
using System.Diagnostics;

namespace CalculatorLibrary
{
    public class Calculator
    {

        public Calculator()
        {
            StreamWriter logFile = File.CreateText("calculator.log");
            Trace.Listeners.Add(new TextWriterTraceListener(logFile));
            Trace.AutoFlush = true;
            Trace.WriteLine("Starting Calculator Log");
            Trace.WriteLine(String.Format("Started {0}",
System.DateTime.Now.ToString()));
        }
```

```
        public double DoOperation(double num1, double num2, string op)
        {
            double result = double.NaN; // Default value is "not-a-number"
if an operation, such as division, could result in an error.

            // Use a switch statement to do the math.
            switch (op)
            {
                case "a":
                    result = num1 + num2;
                    Trace.WriteLine(String.Format("{0} + {1} = {2}", num1,
num2, result));
                    break;
                case "s":
                    result = num1 - num2;
                    Trace.WriteLine(String.Format("{0} - {1} = {2}", num1,
num2, result));
                    break;
                case "m":
                    result = num1 * num2;
                    Trace.WriteLine(String.Format("{0} * {1} = {2}", num1,
num2, result));
                    break;
                case "d":
                    // Ask the user to enter a non-zero divisor.
                    if (num2 != 0)
                    {
                        result = num1 / num2;
                        Trace.WriteLine(String.Format("{0} / {1} = {2}",
num1, num2, result));
                    }
                    break;
                // Return text for an incorrect option entry.
                default:
                    break;
            }
            return result;
        }
    }
}
```

Program.cs 应类似于以下代码：

C#

```
using CalculatorLibrary;

namespace CalculatorProgram
{

    class Program
    {
        static void Main(string[] args)
```

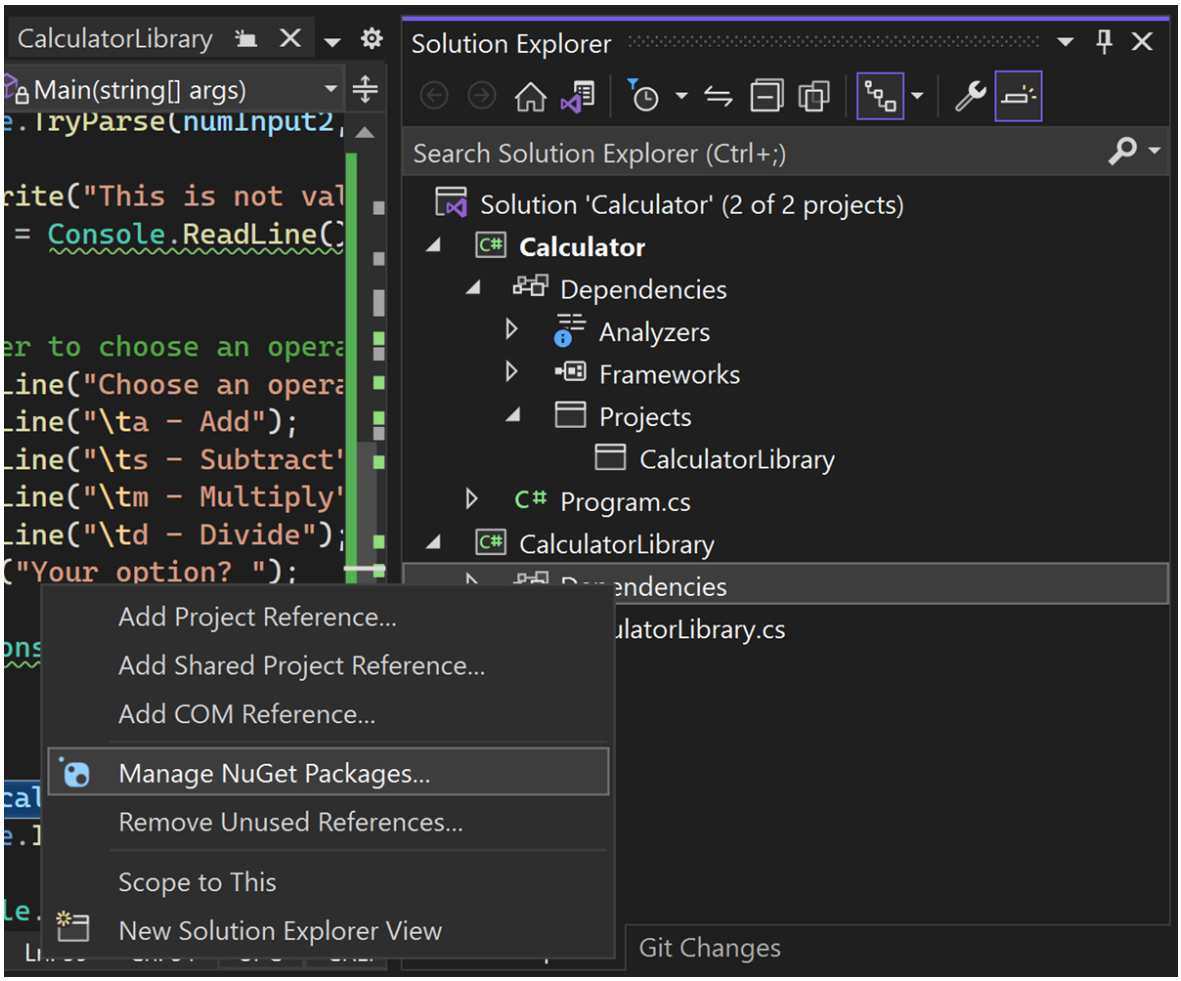
```
        {
            bool endApp = false;
            // Display title as the C# console calculator app.
            Console.WriteLine("Console Calculator in C#\r");
            Console.WriteLine("------------------------\n");

            Calculator calculator = new Calculator();
            while (!endApp)
            {
                // Declare variables and set to empty.
                string numInput1 = "";
                string numInput2 = "";
                double result = 0;

                // Ask the user to type the first number.
                Console.Write("Type a number, and then press Enter: ");
                numInput1 = Console.ReadLine();

                double cleanNum1 = 0;
                while (!double.TryParse(numInput1, out cleanNum1))
                {
                    Console.Write("This is not valid input. Please enter an
integer value: ");
                    numInput1 = Console.ReadLine();
                }

                // Ask the user to type the second number.
                Console.Write("Type another number, and then press Enter:
");
                numInput2 = Console.ReadLine();

                double cleanNum2 = 0;
                while (!double.TryParse(numInput2, out cleanNum2))
                {
                    Console.Write("This is not valid input. Please enter an
integer value: ");
                    numInput2 = Console.ReadLine();
                }

                // Ask the user to choose an operator.
                Console.WriteLine("Choose an operator from the following
list:");
                Console.WriteLine("\ta - Add");
                Console.WriteLine("\ts - Subtract");
                Console.WriteLine("\tm - Multiply");
                Console.WriteLine("\td - Divide");
                Console.Write("Your option? ");

                string op = Console.ReadLine();

                try
                {
                    result = calculator.DoOperation(cleanNum1, cleanNum2,
op);
                    if (double.IsNaN(result))
```

```
                    {
                        Console.WriteLine("This operation will result in a
mathematical error.\n");
                    }
                    else Console.WriteLine("Your result: {0:0.##}\n",
result);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Oh no! An exception occurred trying
to do the math.\n - Details: " + e.Message);
                }

                Console.WriteLine("-----------------------\n");

                // Wait for the user to respond before closing.
                Console.Write("Press 'n' and Enter to close the app, or
press any other key and Enter to continue: ");
                if (Console.ReadLine() == "n") endApp = true;

                Console.WriteLine("\n"); // Friendly linespacing.
            }
            return;
        }
    }
}
```

添加 NuGet 包：写入 JSON 文件

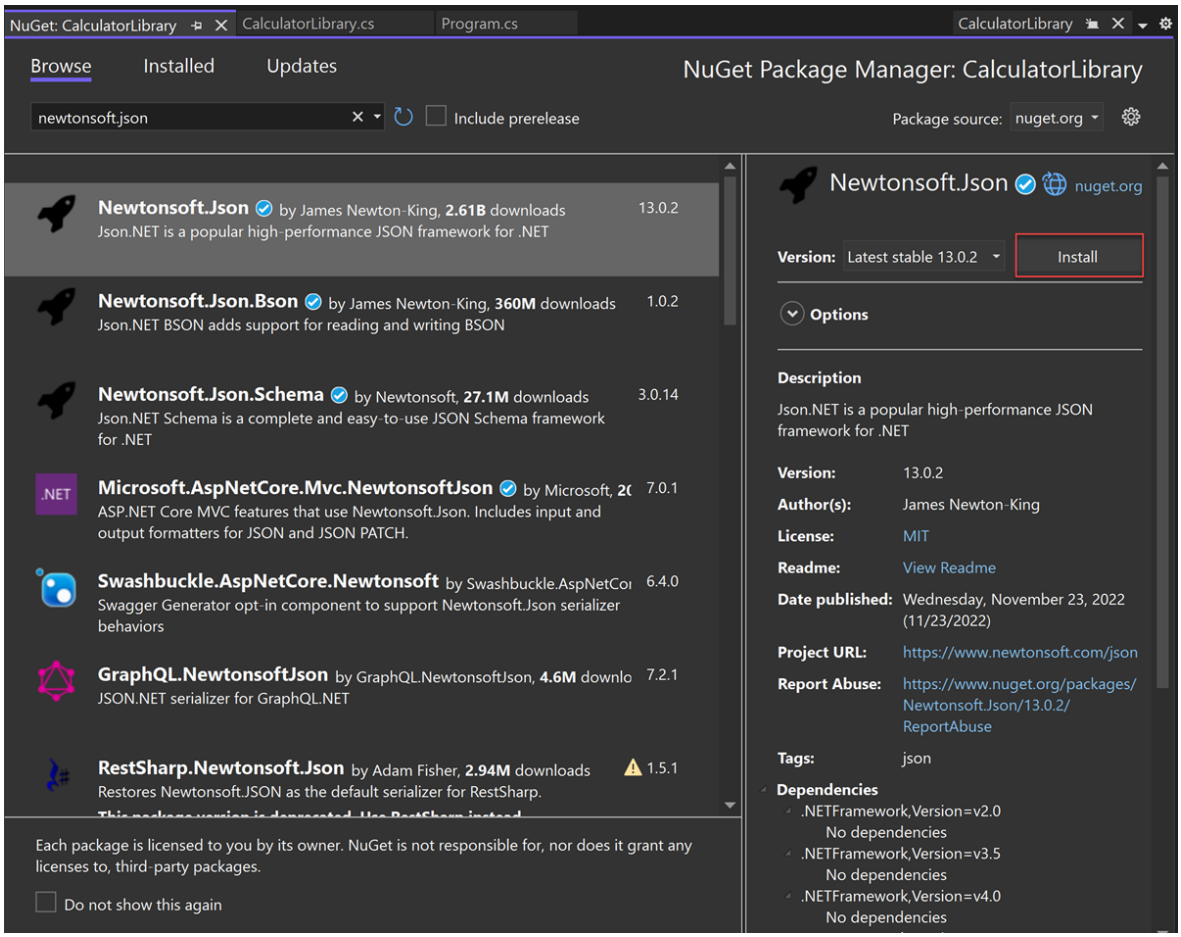
要以 JSON（用于存储对象数据的常用可移植格式）输出操作，可以引用 Newtonsoft.Json NuGet 包。 NuGet 包是 .NET 类库的主要分发方法。

1. 在“解决方案资源管理器”中，右键单击 CalculatorLibrary 项目的“依赖项”节点，然后选择“管理 NuGet 包” 。



“NuGet 包管理器”随即打开。

2. 搜索并选择 Newtonsoft.Json 包，然后选择“安装”。



如果系统提示是否接受更改，请选择“确定”。

Visual Studio 下载包并将其添加到项目。 “解决方案资源管理器”的“包”节点中随即出现一个新条目 。

在 CalculatorLibrary.cs 的开头为 `Newtonsoft.Json` 添加 `using` 指令。

C#

```
using Newtonsoft.Json;
```

3. 创建 `JsonWriter` 成员对象，将 `Calculator` 构造函数替换为以下代码：

C#

```
    JsonWriter writer;

    public Calculator()
    {
        StreamWriter logFile = File.CreateText("calculatorlog.json");
        logFile.AutoFlush = true;
        writer = new JsonTextWriter(logFile);
        writer.Formatting = Formatting.Indented;
        writer.WriteStartObject();
        writer.WritePropertyName("Operations");
```

```
            writer.WriteStartArray();
        }
```

4. 修改 `DoOperation` 方法以添加 JSON `writer` 代码：

C#

```
        public double DoOperation(double num1, double num2, string op)
        {
            double result = double.NaN; // Default value is "not-a-number"
if an operation, such as division, could result in an error.
            writer.WriteStartObject();
            writer.WritePropertyName("Operand1");
            writer.WriteValue(num1);
            writer.WritePropertyName("Operand2");
            writer.WriteValue(num2);
            writer.WritePropertyName("Operation");
            // Use a switch statement to do the math.
            switch (op)
            {
                case "a":
                    result = num1 + num2;
                    writer.WriteValue("Add");
                    break;
                case "s":
                    result = num1 - num2;
                    writer.WriteValue("Subtract");
                    break;
                case "m":
                    result = num1 * num2;
                    writer.WriteValue("Multiply");
                    break;
                case "d":
                    // Ask the user to enter a non-zero divisor.
                    if (num2 != 0)
                    {
                        result = num1 / num2;
                    }
                    writer.WriteValue("Divide");
                    break;
                // Return text for an incorrect option entry.
                default:
                    break;
            }
            writer.WritePropertyName("Result");
            writer.WriteValue(result);
            writer.WriteEndObject();

            return result;
        }
```

5. 用户输完操作数据后，添加一个方法来完成 JSON 语法。

C#

```
public void Finish()
{
    writer.WriteEndArray();
    writer.WriteEndObject();
    writer.Close();
}
```

6. 在 Program.cs 末尾的 `return;` 之前，添加对 `Finish` 的调用：

C#

```
        // Add call to close the JSON writer before return
        calculator.Finish();
        return;
    }
```

7. 生成并运行应用，然后在输入完几个操作后，输入 n 命令关闭该应用。

8. 在“文件资源管理器”中打开 calculatorlog.json 文件。 你应该会看到如下所示的内容：

JSON

```
{
 "Operations": [
     {
     "Operand1": 2.0,
     "Operand2": 3.0,
     "Operation": "Add",
     "Result": 5.0
     },
     {
     "Operand1": 3.0,
     "Operand2": 4.0,
     "Operation": "Multiply",
     "Result": 12.0
     }
 ]
}
```

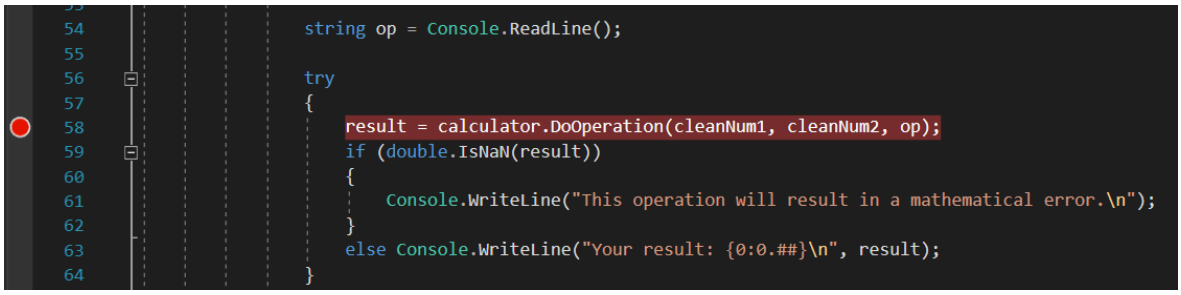
调试：设置并命中断点

Visual Studio 调试器是一个功能强大的工具。 调试器可以单步执行你的代码以找到存在编程错误的确切位置。 然后，你可以了解需要进行哪些更正，并进行临时更改，以便继续运行应用。

1. 在 Program.cs 中，单击以下代码行左侧的装订线。 你也可以在该行中进行单击并选择 F9，或者右键单击并选择“断点”>“插入断点”。

```
C#

result = calculator.DoOperation(cleanNum1, cleanNum2, op);
```

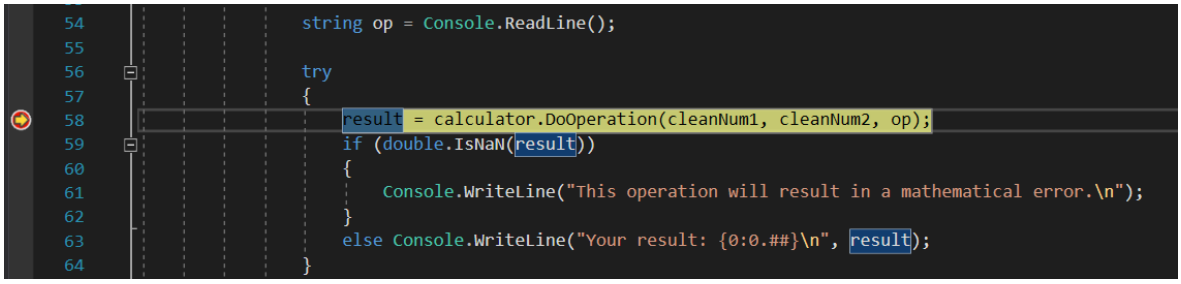
出现的红色圆圈表示断点。 可以使用断点来暂停应用并检查代码。 可以在任意可执行代码行上设置断点。



2. 构建并运行应用。 为计算输入以下值：

   - 对于第一个数字，输入 8。
   - 对于第一个数字，输入 0。
   - 对于运算符，为了增添一些趣味， 我们输入 d。

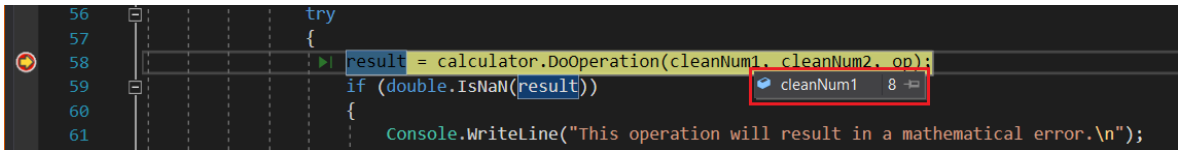
应用在你创建断点的位置（由左侧的黄色指针和突出显示的代码表示）暂停。 突出显示的代码尚未执行。



现在，应用已暂停，你可以检查应用程序状态了。

调试：查看变量

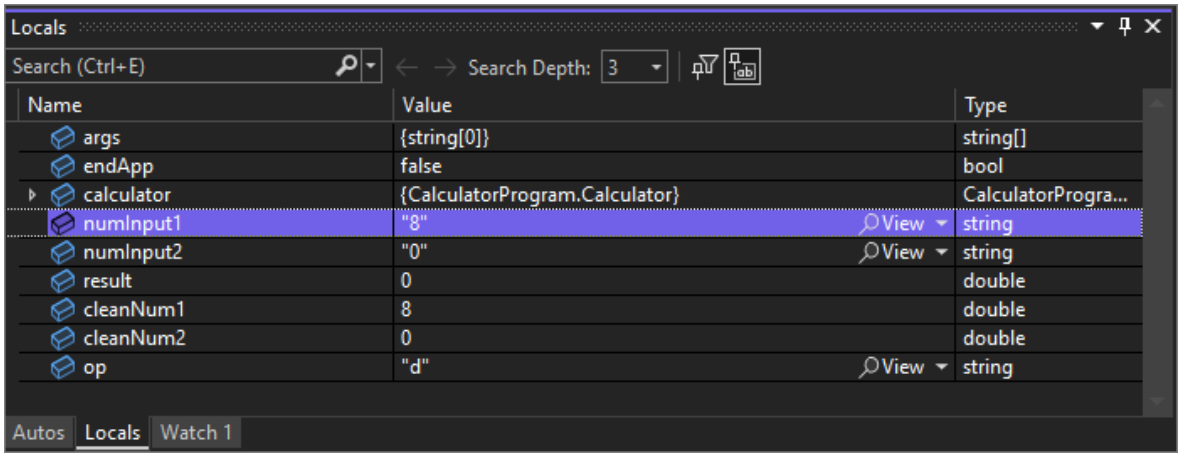
1. 在突出显示的代码中，将鼠标悬停在变量（如 `cleanNum1` 和 `op`）之上。 可以在数据提示中看到这些变量的当前值（分别为 `8` 和 `d`）。



调试时，检查变量是否保留了你希望它们保留的值对于修复问题通常非常关键。

2. 在下面的窗格中，查看“局部变量”窗口。 如果关闭，请依次选择“调试”>“窗口”>“局部变量”将其打开 。

   在“局部变量”窗口中，可以看到当前在范围内的每个变量及其值和类型。



3. 了解一下“自动变量”窗口。

   “自动变量”窗口类似于“局部变量”窗口，不同之处在于它显示应用暂停时所处的当前代码行前后紧跟的变量。

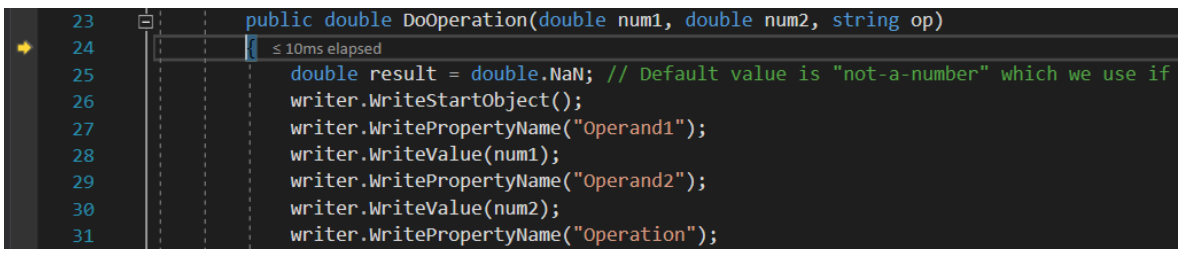
> **备注**
>
> 如果未看到“自动”窗口，请选择“调试”>“Windows”>“自动”将其打开。

接下来，你将在调试器中一次执行一个语句代码，这称为“单步执行”。

调试：单步执行代码

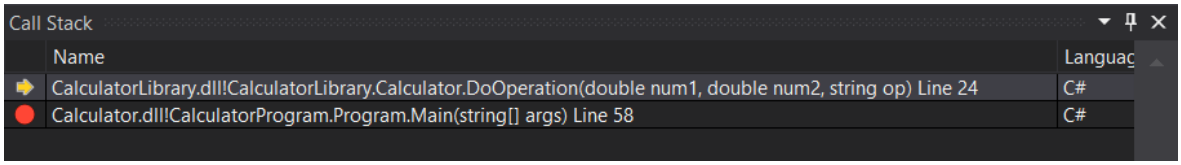
1. 按 F11，或选择“调试”>“单步执行” 。

   如果使用的是“单步执行”命令，应用将在执行当前语句后前进到下一个可执行语句（通常是下一个代码行）。 左侧的黄色指针始终表示当前语句。



   你刚刚单步执行到了 `Calculator` 类中的 `DoOperation` 方法。

2. 若要分层查看程序流，请查看“调用堆栈”窗口。 如果关闭，请依次选择“调试”>“窗口”>“局部变量”将其打开 。



此视图显示当前 `Calculator.DoOperation` 方法，由黄色指针指示。 第二行显示从 Program.cs 中的 `Main` 方法调用该方法的函数。

“调用堆栈”窗口显示方法和函数被调用的顺序 。 此外，通过此窗口还可以访问许多调试器功能问，例如从其快捷菜单访问“转到源代码”。

3. 按 F10（或依次选择“调试”>“不进入子函数”）几次，直到应用在 `switch` 语句处暂停 。

```
C#

switch (op)
{
```

“不进入子函数”命令与“单步执行”命令类似，不同之处在于，如果当前语句调用了函数，则调试器会运行函数中的代码，并且直到函数返回结果时才会暂停执行。 如果你对某个特定函数不感兴趣，“不进入子函数”命令会比“单步执行”快。

4. 再按一次 F10，让应用在以下代码行处暂停。

```
C#

if (num2 != 0)
{
```

此代码检查是否存在除以 0 的情况。 如果应用继续运行，则会引发一般性异常（错误），但你可能想要执行其他操作，例如在控制台中查看实际返回值。 一种方法是使用称为“编辑并继续”的调试器功能对代码进行更改，然后继续调试。 但是，可以使用其他技巧来临时修改执行流。

调试：测试临时更改

1. 选择当前在 `if (num2 != 0)` 语句处暂停的黄色指针，然后将它拖到下面的语句：

```
C#
```

```
result = num1 / num2;
```

将指针拖到此处会使应用完全跳过 `if` 语句，让你可以看到除以零时返回的结果。

2. 按 F10 执行代码行。
3. 如果将鼠标悬停在 `result` 变量上，则会显示“无穷大”值。 在 C# 中，“无穷大”是除以 0 的结果。
4. 按 F5，或依次选择“调试”>“继续调试” 。

   无穷大符号作为数学运算的结果显示在控制台中。
5. 输入 n 命令正确关闭应用。

代码完成

下面是完成所有步骤后 CalculatorLibrary.cs 文件的完整代码：

C#

```
using Newtonsoft.Json;

namespace CalculatorLibrary
{
    public class Calculator
    {

        JsonWriter writer;

        public Calculator()
        {
            StreamWriter logFile = File.CreateText("calculatorlog.json");
            logFile.AutoFlush = true;
            writer = new JsonTextWriter(logFile);
            writer.Formatting = Formatting.Indented;
            writer.WriteStartObject();
            writer.WritePropertyName("Operations");
            writer.WriteStartArray();
        }

        public double DoOperation(double num1, double num2, string op)
        {
            double result = double.NaN; // Default value is "not-a-number"
if an operation, such as division, could result in an error.
            writer.WriteStartObject();
            writer.WritePropertyName("Operand1");
            writer.WriteValue(num1);
            writer.WritePropertyName("Operand2");
```

```
            writer.WriteValue(num2);
            writer.WritePropertyName("Operation");
            // Use a switch statement to do the math.
            switch (op)
            {
                case "a":
                    result = num1 + num2;
                    writer.WriteValue("Add");
                    break;
                case "s":
                    result = num1 - num2;
                    writer.WriteValue("Subtract");
                    break;
                case "m":
                    result = num1 * num2;
                    writer.WriteValue("Multiply");
                    break;
                case "d":
                    // Ask the user to enter a non-zero divisor.
                    if (num2 != 0)
                    {
                        result = num1 / num2;
                    }
                    writer.WriteValue("Divide");
                    break;
                // Return text for an incorrect option entry.
                default:
                    break;
            }
            writer.WritePropertyName("Result");
            writer.WriteValue(result);
            writer.WriteEndObject();

            return result;
        }

        public void Finish()
        {
            writer.WriteEndArray();
            writer.WriteEndObject();
            writer.Close();
        }
    }
}
```

下面是 Program.cs 的代码：

C#

```
using CalculatorLibrary;

namespace CalculatorProgram
{
```

```
    class Program
    {
        static void Main(string[] args)
        {
            bool endApp = false;
            // Display title as the C# console calculator app.
            Console.WriteLine("Console Calculator in C#\r");
            Console.WriteLine("------------------------\n");

            Calculator calculator = new Calculator();
            while (!endApp)
            {
                // Declare variables and set to empty.
                string numInput1 = "";
                string numInput2 = "";
                double result = 0;

                // Ask the user to type the first number.
                Console.Write("Type a number, and then press Enter: ");
                numInput1 = Console.ReadLine();

                double cleanNum1 = 0;
                while (!double.TryParse(numInput1, out cleanNum1))
                {
                    Console.Write("This is not valid input. Please enter an
integer value: ");
                    numInput1 = Console.ReadLine();
                }

                // Ask the user to type the second number.
                Console.Write("Type another number, and then press Enter:
");
                numInput2 = Console.ReadLine();

                double cleanNum2 = 0;
                while (!double.TryParse(numInput2, out cleanNum2))
                {
                    Console.Write("This is not valid input. Please enter an
integer value: ");
                    numInput2 = Console.ReadLine();
                }

                // Ask the user to choose an operator.
                Console.WriteLine("Choose an operator from the following
list:");
                Console.WriteLine("\ta - Add");
                Console.WriteLine("\ts - Subtract");
                Console.WriteLine("\tm - Multiply");
                Console.WriteLine("\td - Divide");
                Console.Write("Your option? ");

                string op = Console.ReadLine();

                try
```

```
                {
                    result = calculator.DoOperation(cleanNum1, cleanNum2,
op);
                    if (double.IsNaN(result))
                    {
                        Console.WriteLine("This operation will result in a
mathematical error.\n");
                    }
                    else Console.WriteLine("Your result: {0:0.##}\n",
result);
                }
                catch (Exception e)
                {
                    Console.WriteLine("Oh no! An exception occurred trying
to do the math.\n - Details: " + e.Message);
                }

                Console.WriteLine("-----------------------\n");

                // Wait for the user to respond before closing.
                Console.Write("Press 'n' and Enter to close the app, or
press any other key and Enter to continue: ");
                if (Console.ReadLine() == "n") endApp = true;

                Console.WriteLine("\n"); // Friendly linespacing.
            }
            calculator.Finish();
            return;
        }
    }
}
```

后续步骤

恭喜你完成本教程！ 要更加深入地了解，请继续学习以下内容：

- 继续学习更多 C# 教程
- 快速入门：创建 ASP.NET Core Web 应用
- 了解如何在 Visual Studio 中调试 C# 代码
- 演练如何创建和运行单元测试
- 运行 C# 程序
- 了解 C# IntelliSense
- 继续学习 Visual Studio IDE 概述
- 日志记录和跟踪

教程：Visual Studio 中的 C# 和 ASP.NET Core 入门

项目 • 2023/11/23

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

本教程介绍借助 ASP.NET Core 进行 C# 开发，在 Visual Studio 中创建一个 C# ASP.NET Core Web 应用。

本教程演示如何：

- ✓ 创建 Visual Studio 项目
- ✓ 创建 C# ASP.NET Core Web 应用
- ✓ 对 Web 应用进行更改
- ✓ 探索 IDE 功能
- ✓ 运行 Web 应用

先决条件

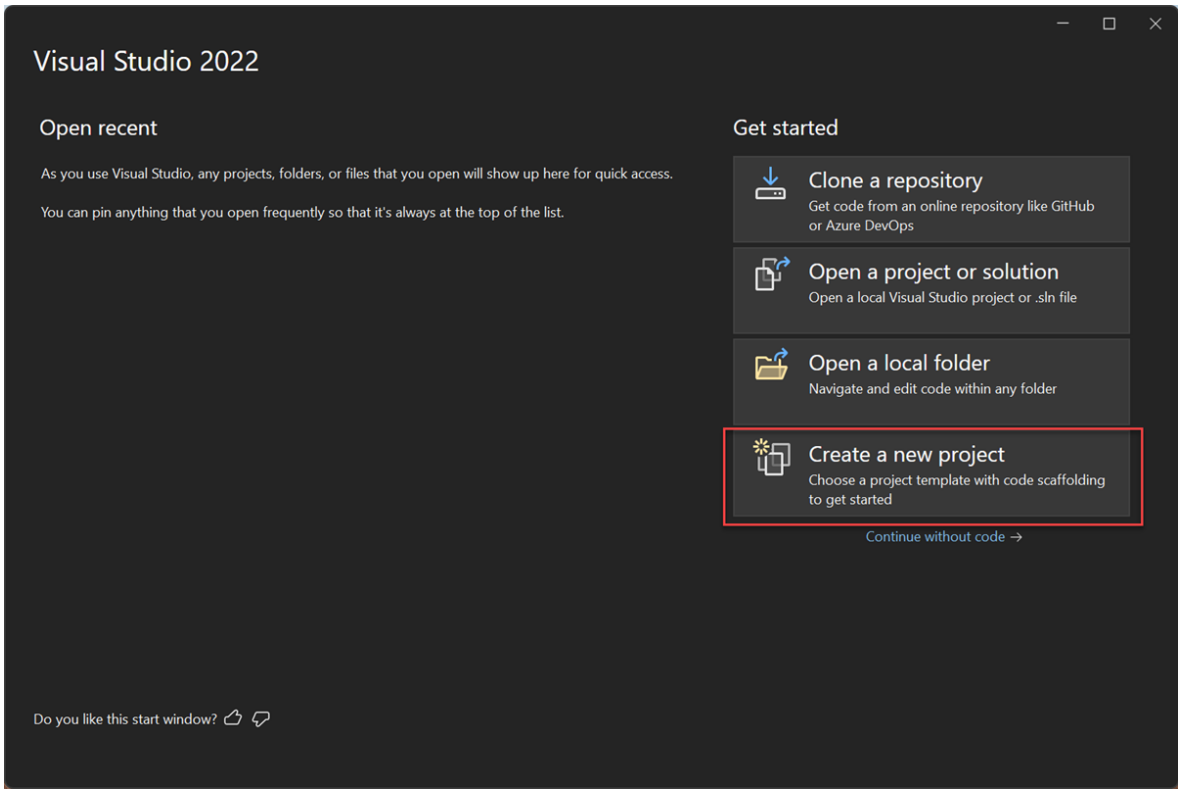
若要完成本教程，必须具有 Visual Studio。 请访问 Visual Studio 下载页 获取免费版本。

- 有关升级到最新 Visual Studio 版本的详细信息，请参阅 Visual Studio更新。
- 若要自定义 Visual Studio 体验，请参阅个性化 Visual Studio IDE 和编辑器。

创建项目

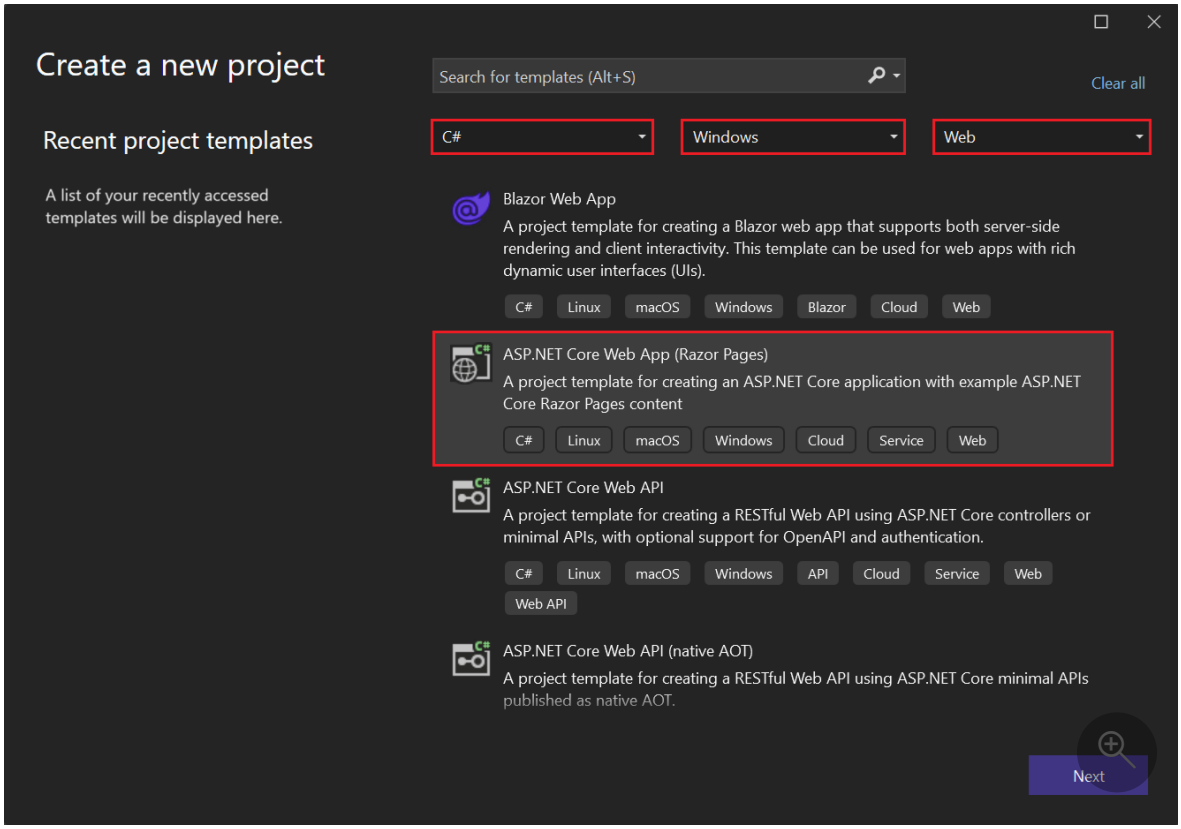
首先，创建一个 ASP.NET Core 项目。 项目类型随附了构建功能完备的网站所需的全部模板文件。

1. 在“开始”窗口中，选择“创建新项目”。



2. 在“创建新项目”窗口中，从“语言”列表中选择“C#”。 接下来，从“平台”列表中选择“Windows”，然后从“项目类型”列表中选择“Web”。

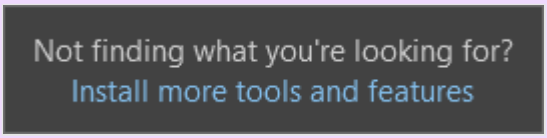
   应用语言、平台和项目类型筛选器之后，选择“ASP.NET Core Web 应用”模板，然后选择“下一步”。



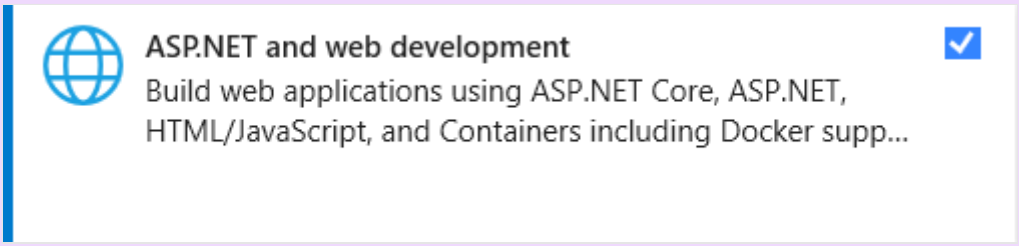
> **① 备注**

如果未看到“ASP.NET Core Web 应用”模板，则可以从“创建新项目”窗口安装该模板。

在模板列表底部的“未找到你要查找的内容?”消息中，选择“安装更多工具和功能”链接。



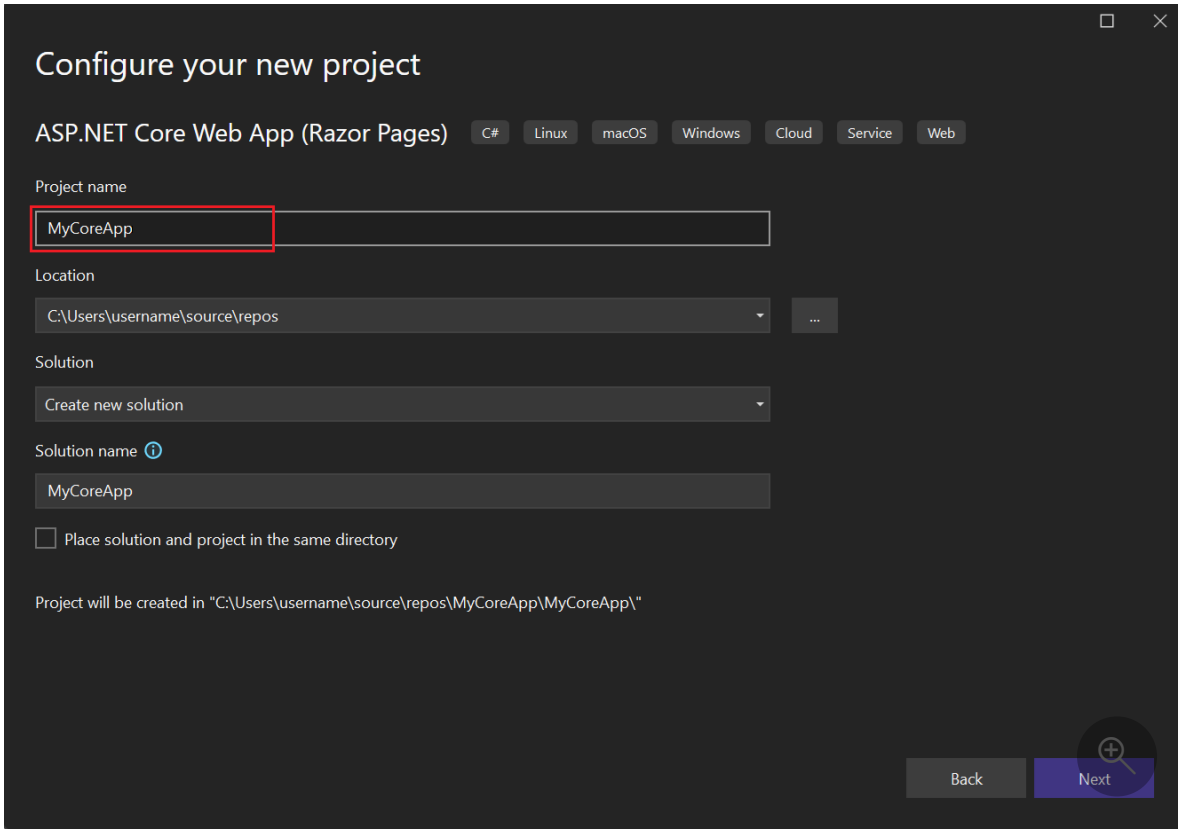
在 Visual Studio 安装程序中，选择“ASP.NET 和 Web 开发”工作负载。



在 Visual Studio 安装程序中，选择“修改”。 系统可能会提示你保存工作内容。 选择“继续”以安装工作负载。

返回到“**创建项目**”过程中的步骤 2。

3. 在“配置新项目”窗口中，在“项目名称”字段中输入“MyCoreApp”。 然后，选择“下一步”。

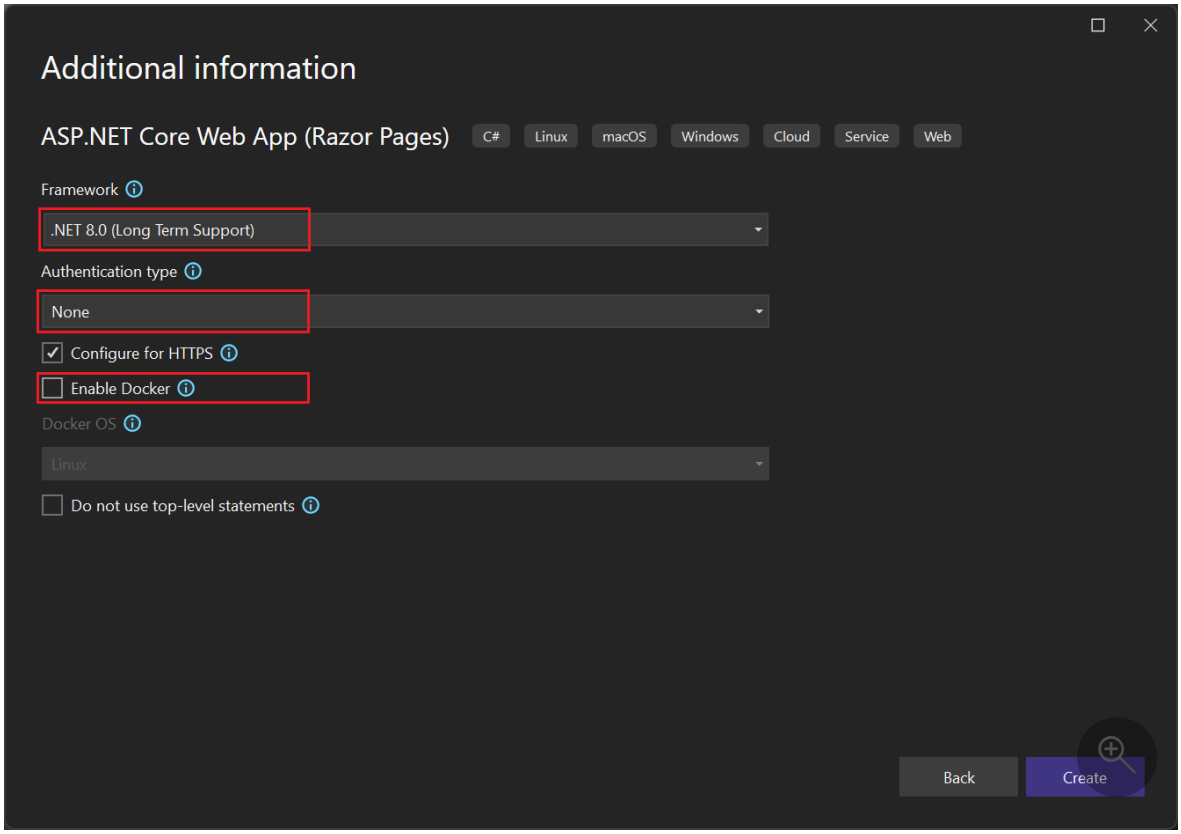


4. 在“其他信息”窗口中，验证“目标框架”字段中是否显示了“.NET 8.0”。

   在此窗口中，可以启用 Docker 支持并添加身份验证支持。 “身份验证类型”的下拉菜单具有以下四个选项：

   - **无**：不进行身份验证。
   - **个人帐户**：这些身份验证存储在本地或基于 Azure 的数据库中。
   - **Microsoft 标识平台**：此选项使用 Microsoft Entra ID 或 Microsoft 365 进行身份验证。
   - **Windows**：适用于 Intranet 应用程序。

   保持“启用 Docker”框处于未选中状态，并选择“无”作为身份验证类型。



5. 选择**创建**。

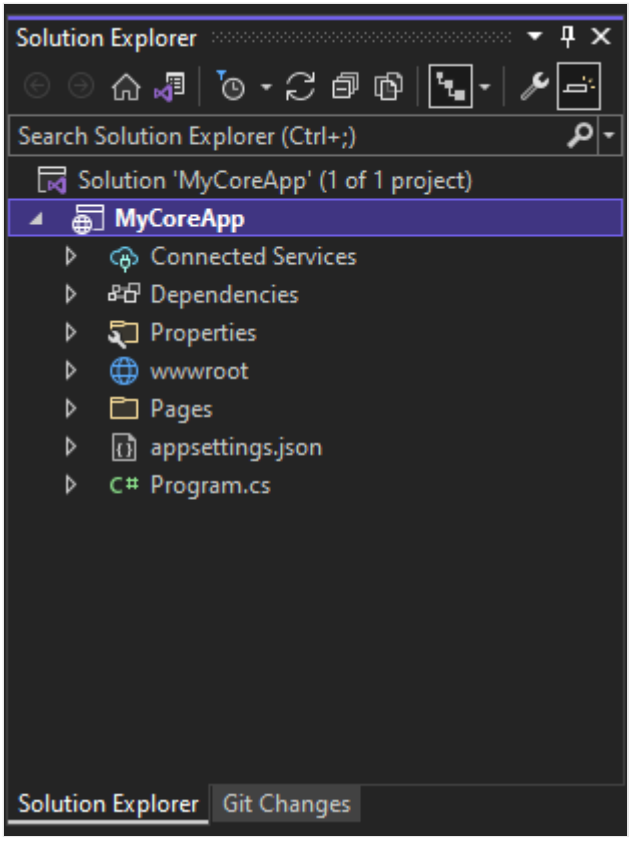
此时，Visual Studio 将打开新项目。

关于解决方案

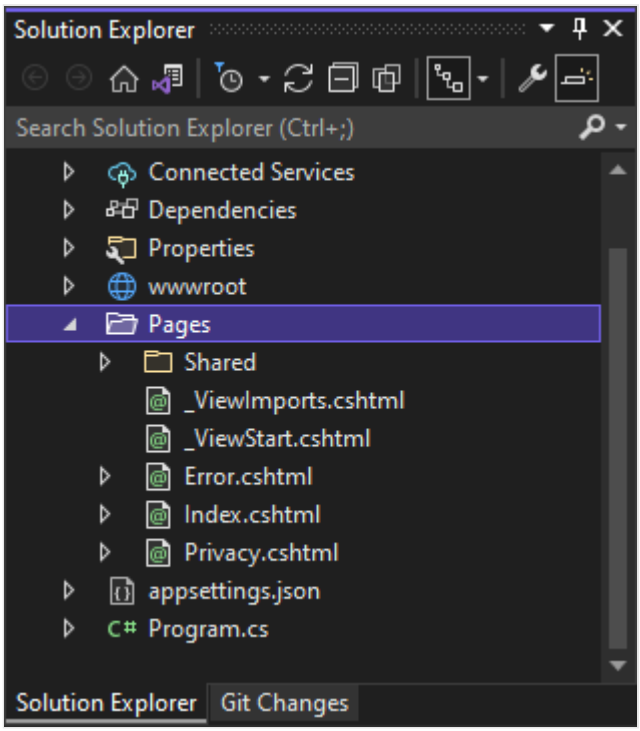
此解决方案遵循 Razor 页面 设计模式。 它与 Model-View-Controller (MVC) 设计模式的不同之处在于，它进行了优化，以包含 Razor Page 本身的模型和控制器代码。

浏览解决方案

1. 项目模板会创建一个解决方案，其中包含一个名为 MyCoreApp 的 ASP.NET Core 项目。 选择“解决方案资源管理器”选项卡以查看其内容。



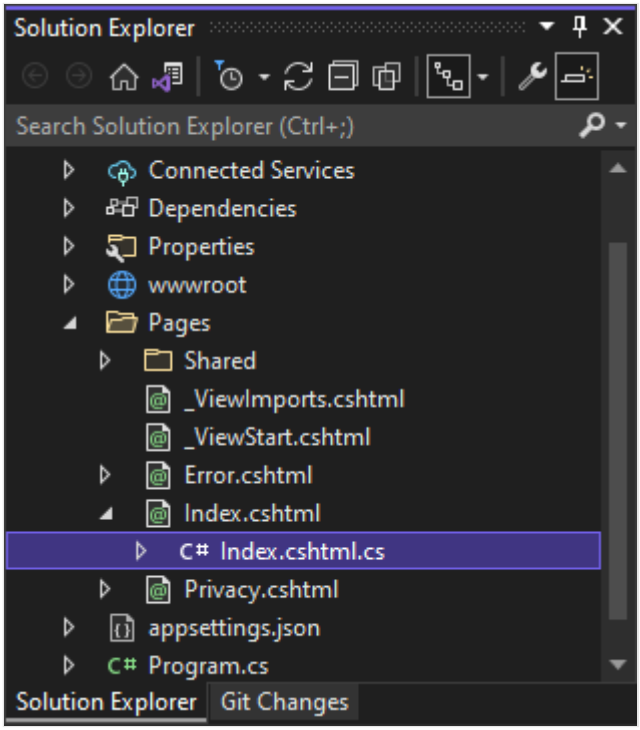
2. 展开“页面”文件夹 。



3. 选择“Index.cshtml”文件，并在代码编辑器中查看。

```
Index.cshtml  MyCoreApp: Overview
@page
@model IndexModel
@{
    ViewData["Title"] = "Home page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://docs.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
</div>
```

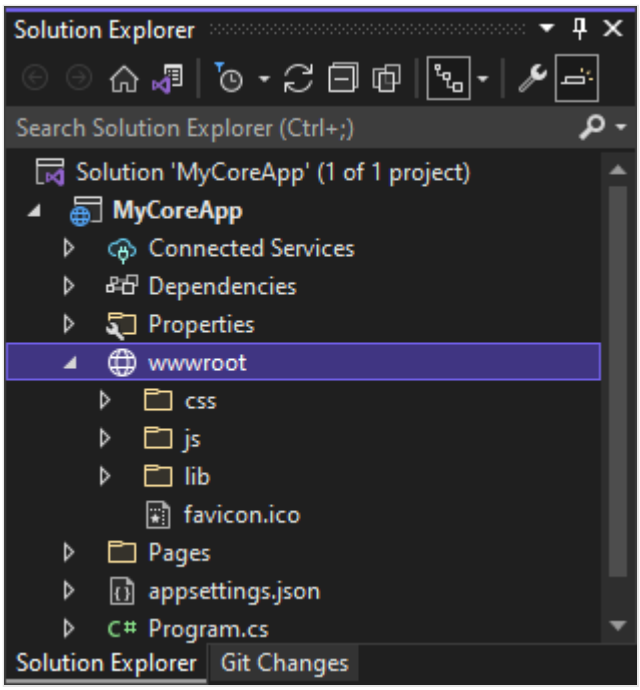
4. 每个 .cshtml 文件都具有关联的代码文件。 若要在编辑器中打开代码文件，请在“解决方案资源管理器”中展开“Index.cshtml”节点，并选择“Index.cshtml.cs”文件。



5. 查看代码编辑器中的“Index.cshtml.cs”文件 。

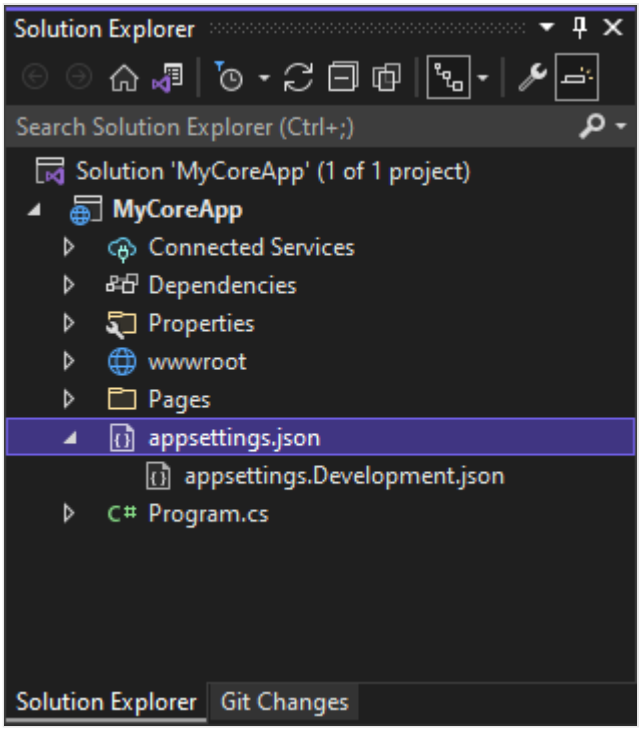
```
Index.cshtml.cs  Index.cshtml  MyCoreApp: Overview
MyCoreApp  MyCoreApp.Pages.IndexModel  _logger
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace MyCoreApp.Pages
{
    8 references
    public class IndexModel : PageModel
    {
        private readonly ILogger<IndexModel> _logger;

        0 references
        public IndexModel(ILogger<IndexModel> logger)
        {
            _logger = logger;
        }

        0 references
        public void OnGet()
        {

        }
    }
}
```

6. 该项目包含一个 wwwroot 文件夹，它是网站的根。 展开文件夹以查看其内容。



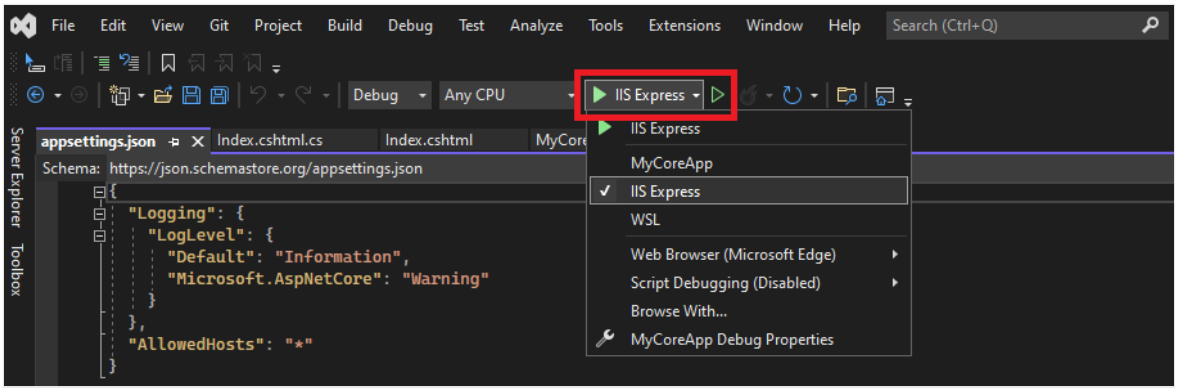
可将静态站点内容（例如 CSS、图像和 JavaScript 库）直接放置在所需的路径中。

7. 该项目还包含在运行时管理 Web 应用的配置文件。 默认应用程序配置存储在 appsettings.json 中。 但是，可使用 appsettings.Development.json 替代这些设置。 展开 appsettings.json 文件以查看 appsettings.Development.json 文件 。



运行、调试和更改

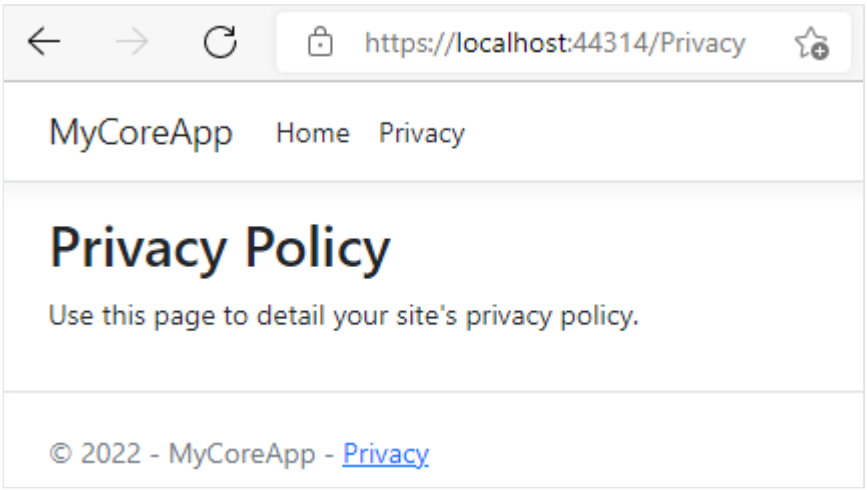
1. 在工具栏中选择“IIS Express”按钮，在调试模式下生成并运行应用。 或者，按 F5 或从菜单栏转到“调试”>“启动调试”。



> **备注**
>
> 如果收到错误消息“无法连接到 Web 服务器 'IIS Express'”，请关闭 Visual Studio，然后以管理员身份重新启动程序。 要执行此任务，可右键单击“开始”菜单中的 Visual Studio 图标，然后从上下文菜单中选择“以管理员身份运行”选项。
>
> 系统可能会向你发送一条消息，询问你是否接受 IIS SSL Express 证书。 要在 Web 浏览器中查看代码，请选择“是”，如果收到后续安全警告消息，也请选择“是”。

2. Visual Studio 启动浏览器窗口。 然后，应在菜单栏中看到“主页”和“隐私”页面。

3. 从菜单栏中选择“隐私”。 浏览器中的“隐私”页呈现在 Privacy.cshtml 文件中设置的文本。



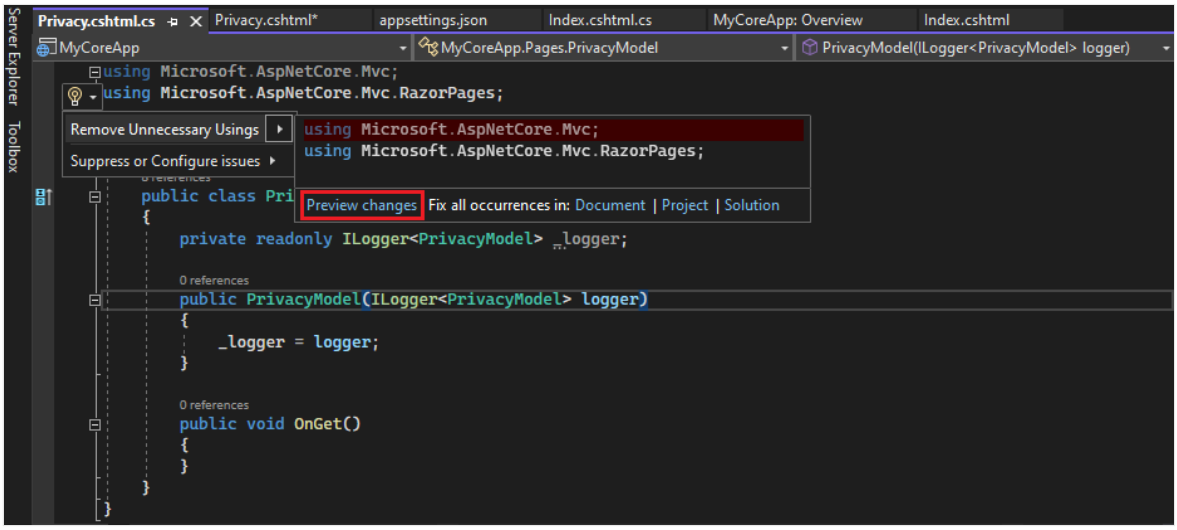
4. 返回到 Visual Studio，然后按 Shift+F5 停止调试。 此操作会关闭浏览器窗口中的项目。

5. 在 Visual Studio 中，打开要编辑的“Privacy.cshtml” 。 接下来，删除“使用此页面详细说明网站的隐私策略”语句，并将其替换为“此页面自 @ViewData["TimeStamp"] 起构建”。

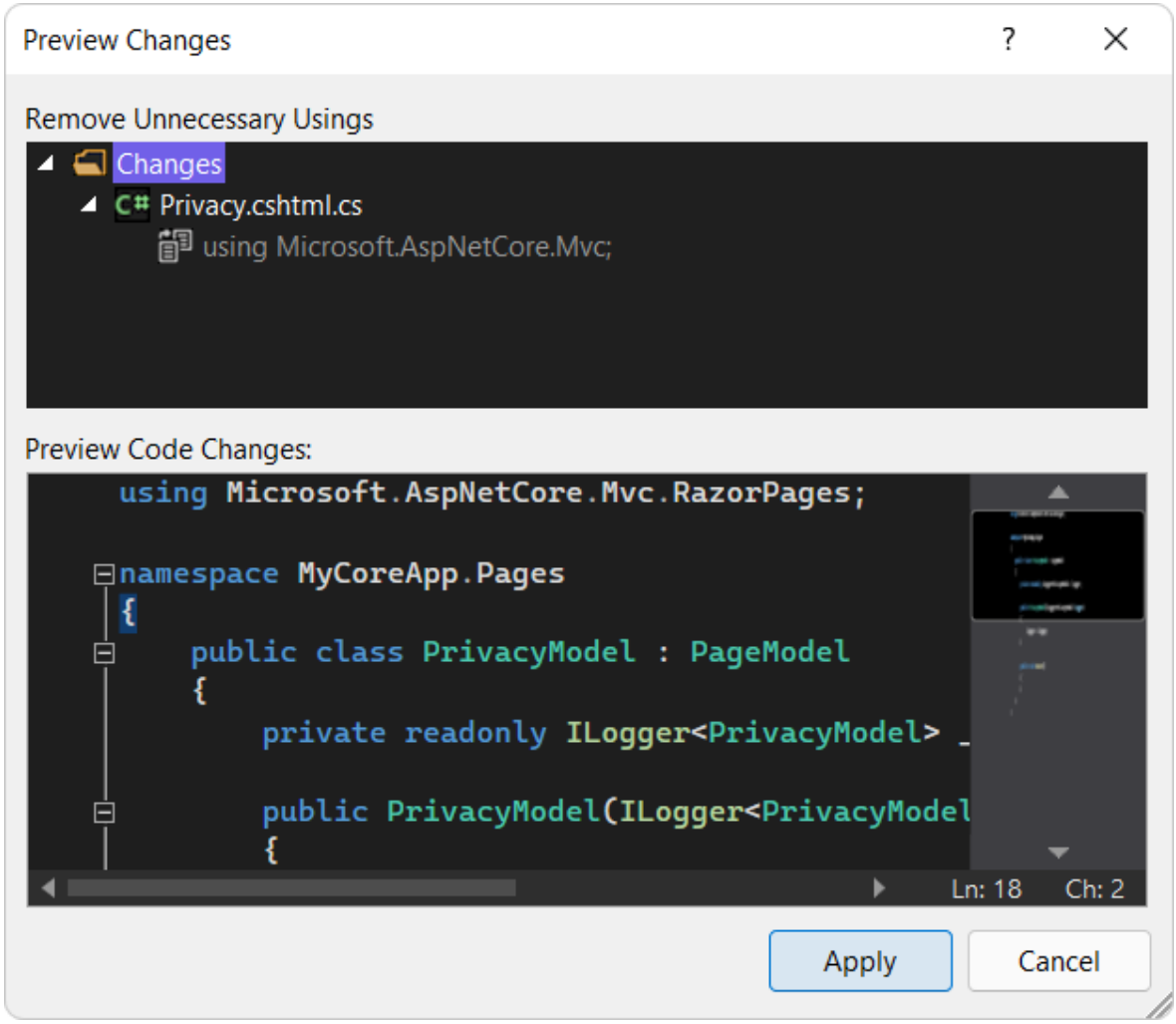
```
@page
@model PrivacyModel
@{
    ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>

<p>This page is under construction as of @ViewData["Timestamp"]</p>
```

6. 现在，让我们对代码进行更改。 选择“Privacy.cshtml.cs”。 然后，选择以下快捷方式清理文件顶部的 `using` 指令：

   鼠标悬停或选择灰显的 `using` 指令。 快速操作灯泡将出现在插入点下方或左边距中。 选择灯泡，然后选择“移除不必要的使用”旁边的“展开”箭头。



   现在，选择“预览更改”来查看所更改的内容。



选择"应用"。 Visual Studio 从文件中删除不必要的 `using` 指令。

7. 接下来，使用 DateTime.ToString 方法为针对区域性或区域设置格式的当前日期创建字符串。

   - 该方法的第一个参数指定如何显示日期。 此示例使用格式说明符 (`d`) 指示短日期格式。
   - 第二个参数是 CultureInfo 对象，该对象指定日期的区域性或区域。 第二个参数确定日期中任意单词的语言以及所使用的分隔符类型等。

   将 `OnGet()` 方法的主体更改为以下代码：

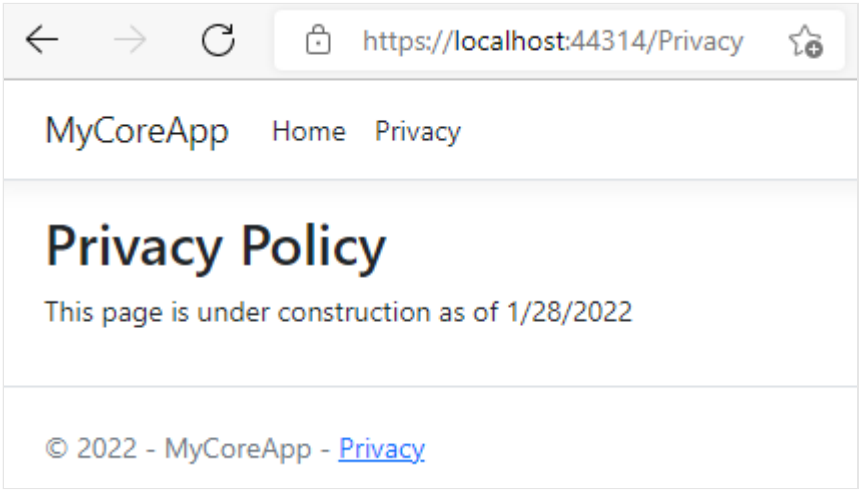
```C#
public void OnGet()
{
   string dateTime = DateTime.Now.ToString("d", new CultureInfo("en-US"));
   ViewData["TimeStamp"] = dateTime;
}
```

8. 请注意，以下 `using` 指令会自动添加到文件顶部：

```
C#

using System.Globalization;
```

`System.Globalization` 包含 CultureInfo 类。

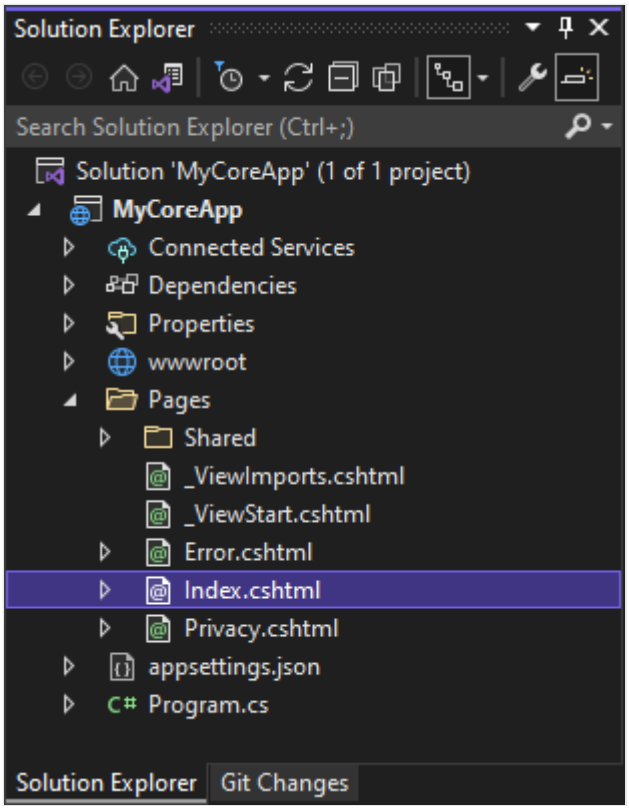
9. 按 F5，即可在 Web 浏览器中打开项目 。
10. 在网站顶部，选择“隐私”查看你的更改。



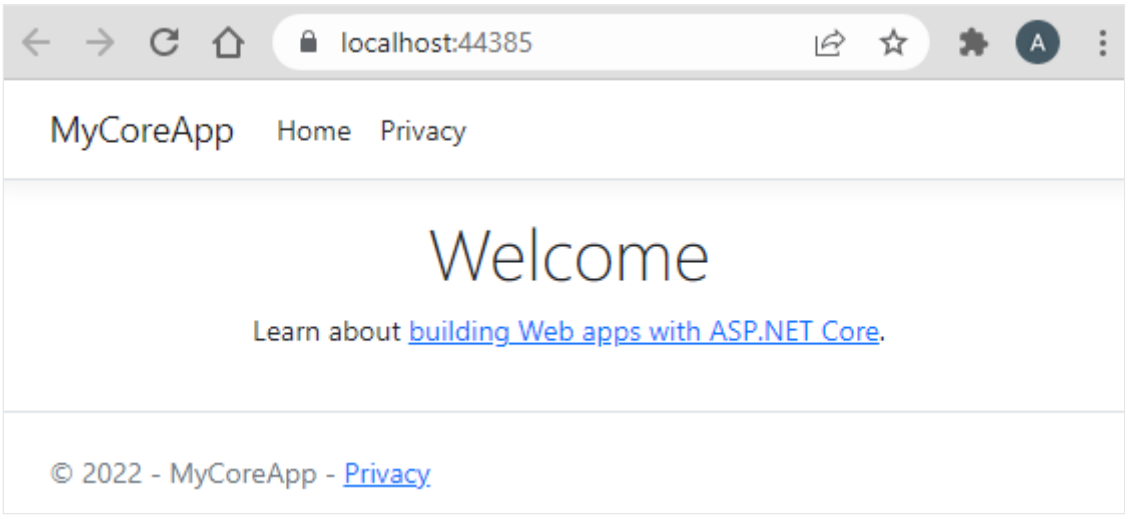
11. 关闭 Web 浏览器，按 Shift+F5 停止调试。

更改你的主页

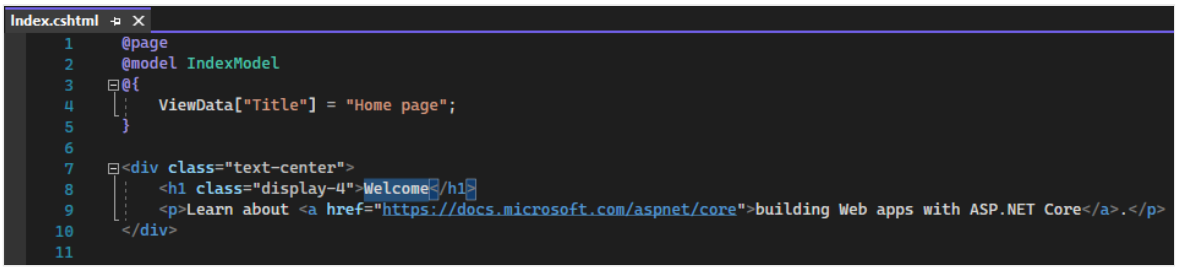
1. 在“解决方案资源管理器”中，展开“Pages”文件夹，然后选择“Index.cshtml”。



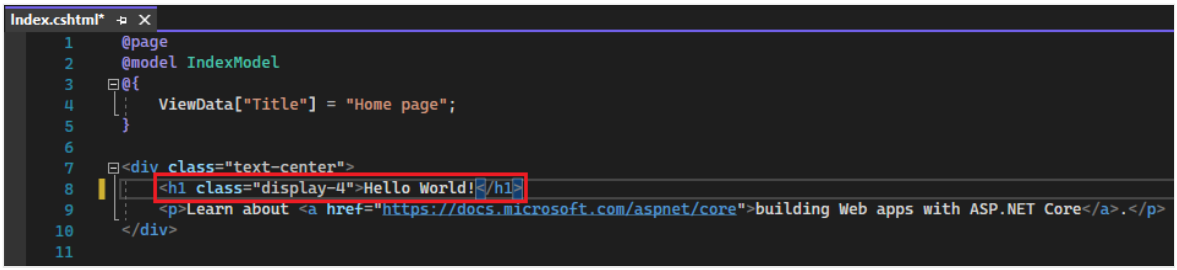
“Index.cshtml”文件对应于在 Web 浏览器中运行的 Web 应用中的“主页”。



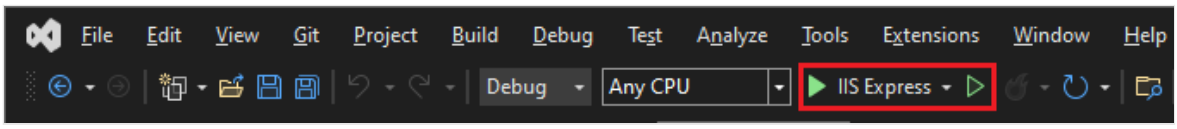
在代码编辑器，可以看到主页上显示的文本的 HTML 代码。



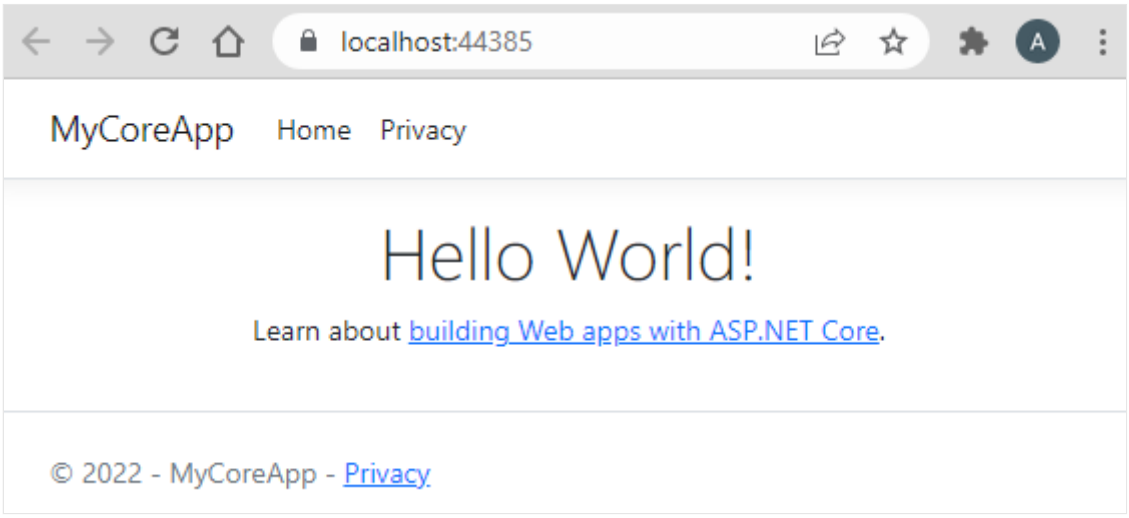
2. 将“欢迎”文本替换为“Hello World!”



3. 选择“IIS Express”或按 Ctrl+F5 运行此应用，并在 Web 浏览器中打开它 。



4. 在 Web 浏览器中，可以在主页上看到新更改。



5. 关闭 Web 浏览器，按 Shift+F5 停止调试，并保存项目。 现在可以关闭 Visual Studio。

后续步骤

恭喜你完成本教程！ 希望你在学习 C#、ASP.NET Core 和 Visual Studio IDE 时乐享其中。 要详细了解如何使用 C# 和 ASP.NET 创建 Web 应用或网站，请继续学习以下教程：

使用 ASP.NET Core 创建 Razor 页面 Web 应用

或者，了解如何使用 Docker 容器化 Web 应用：

Visual Studio 中的容器工具

另请参阅

使用 Visual Studio 将 Web 应用发布到 Azure App Service

教程：在 Visual Studio 中使用 XAML 和 C# 创建第一个 Windows App SDK 应用程序

项目 • 2024/01/13

在这个 Visual Studio 集成开发环境 (IDE) 简介中，你将创建能在任何 Windows 10 或更高版本设备上运行的“Hello World”应用。 为此，将使用 Windows App SDK (WinUI 3) 项目模板、Extensible Application Markup Language (XAML) 和 C# 编程语言。

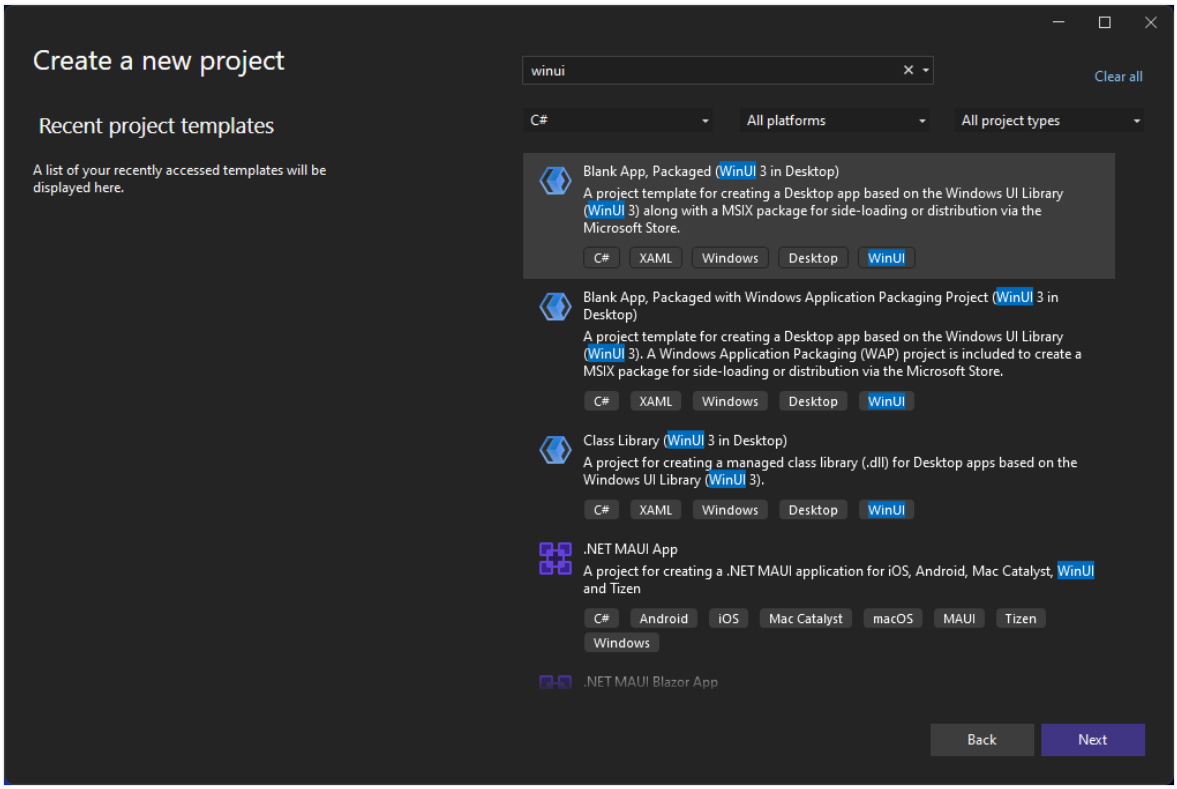
> **备注**
>
> WinUI 3 是本机 UI 平台组件，随附 **Windows 应用 SDK**（与 **Windows SDK** 完全分离）。 有关详细信息，请参阅 **WinUI 3**。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

创建项目

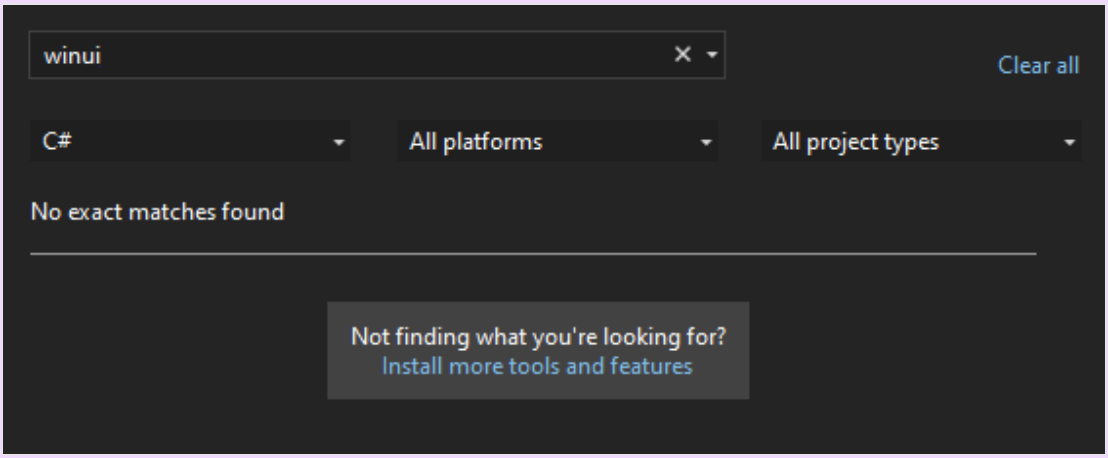
首先，创建一个 WinUI 3 项目。 项目类型随附了所需的全部模板文件，无需添加任何内容！

1. 打开 Visual Studio，然后在“启动”窗口上，选择“创建新项目”。
2. 在“创建新项目”屏幕上，在搜索框中输入“WinUI”，选择“打包的空白应用(桌面版 WinUI 3)”对应的 C# 模板，然后选择“下一步”。

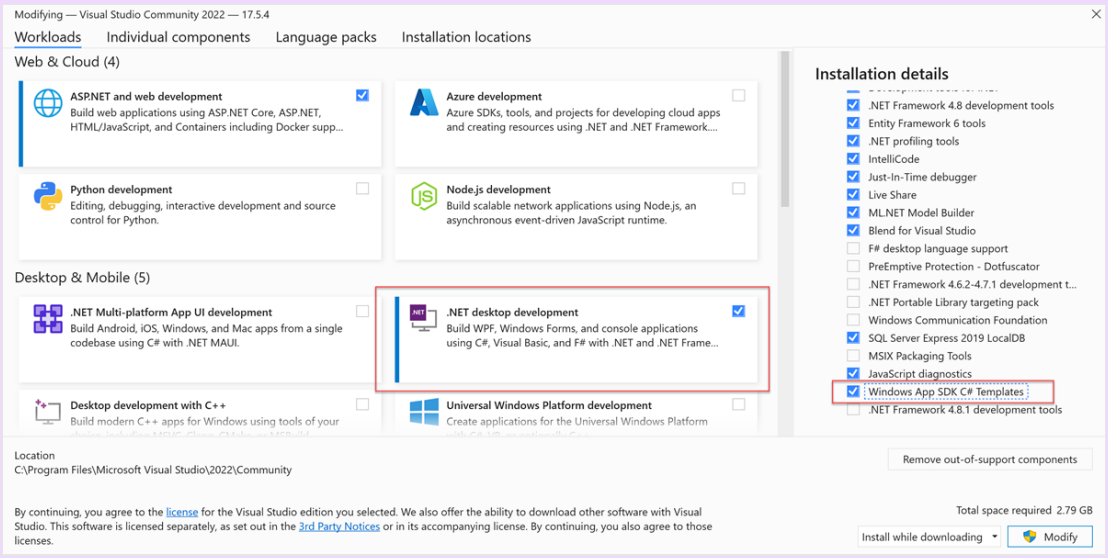


ⓘ 备注

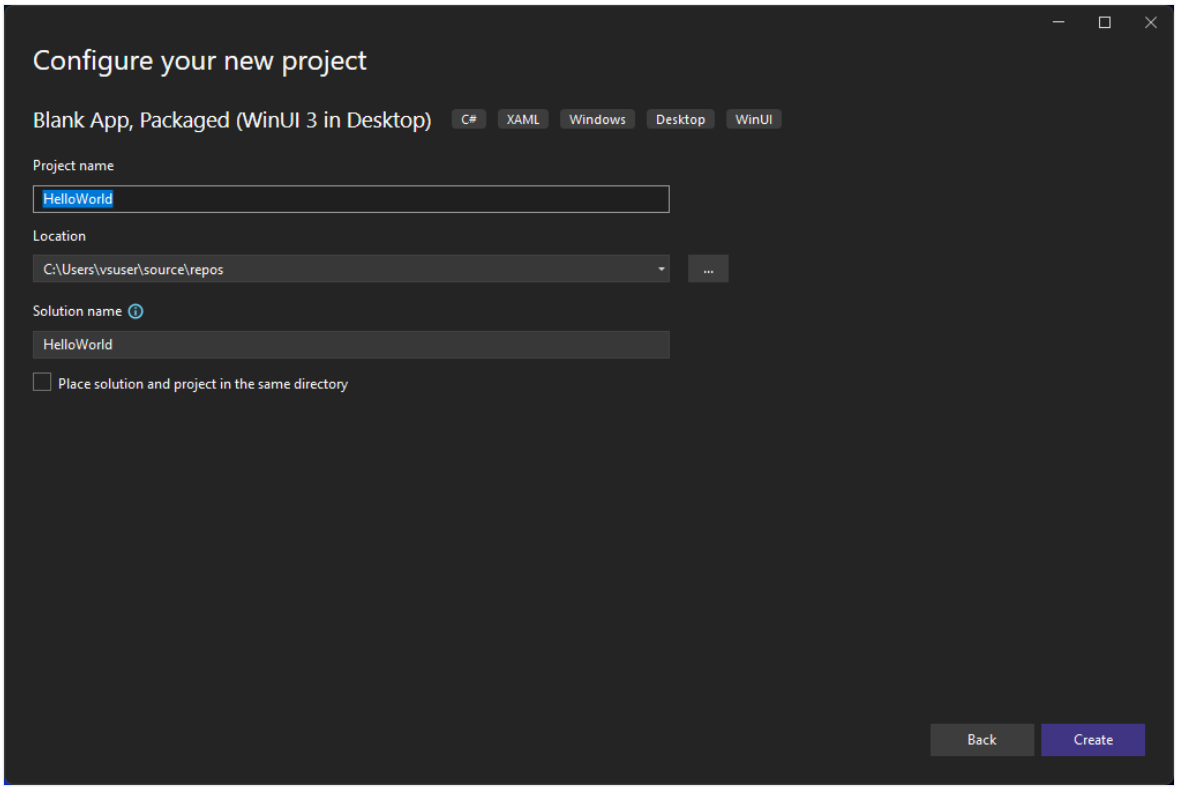
如果没有看到“打包的空白应用(桌面版 WinUI 3)”项目模板，请单击“安装多个工具和功能”链接。



Visual Studio 安装程序启动。 选择“.NET 桌面开发”工作负载，然后在安装对话框的“安装详细信息”窗格中，选择“Windows App SDK C# 模板”（位于列表底部）。 现在选择“修改”。

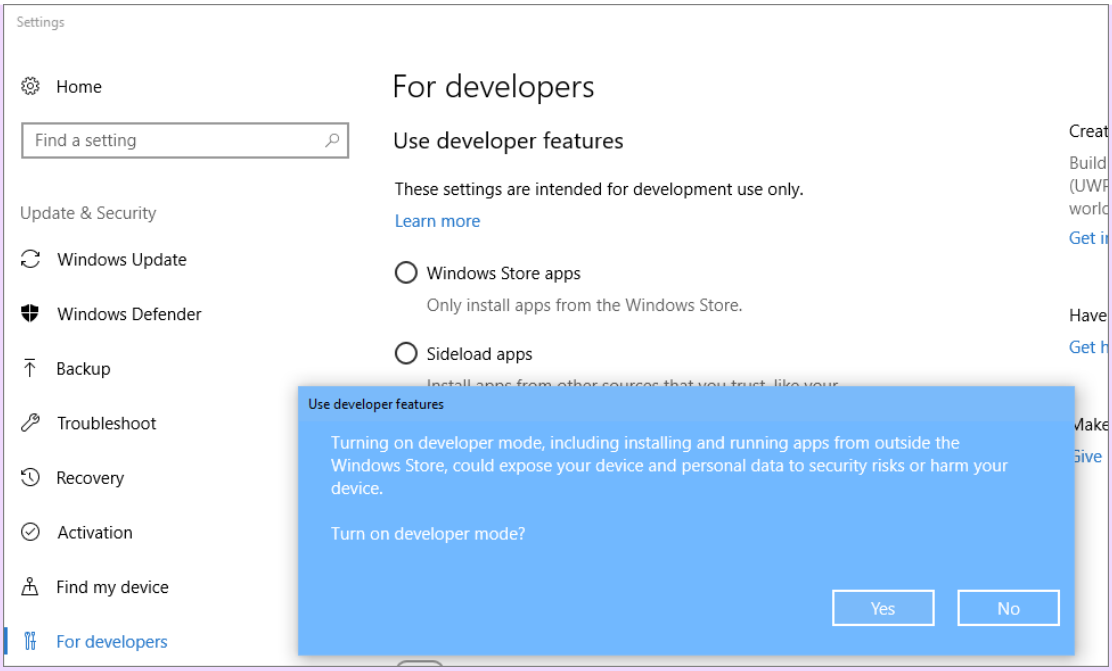


3. 为项目提供名称“HelloWorld”，然后选择“创建”。



ⓘ 备注

如果你是第一次使用 Visual Studio 创建 Windows App SDK 应用，则可能会出现“设置”对话框。 选择“开发人员模式”，然后选择“是”。



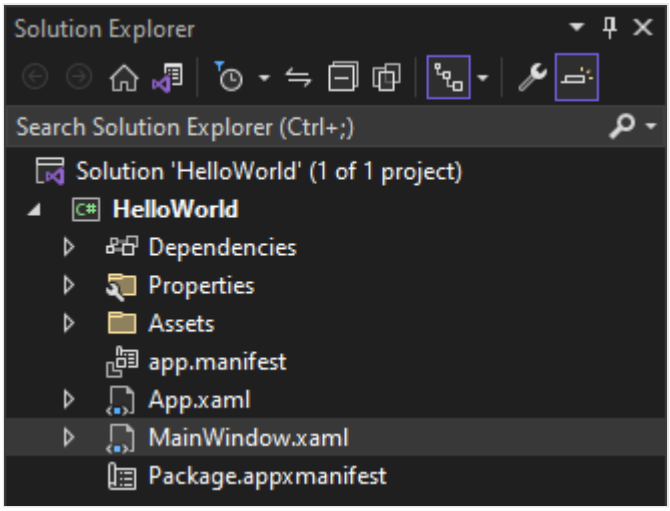
Visual Studio 会为你安装其他开发人员模式包。 包安装完成时，请关闭“设置”对话框 。

创建应用程序

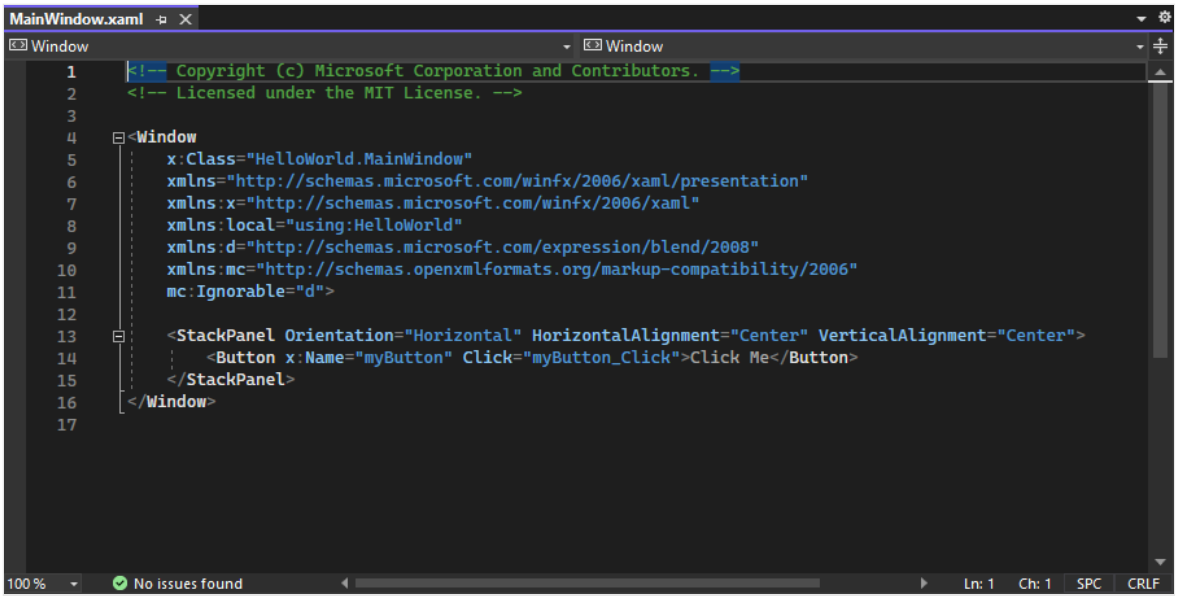
现在开始开发吧。 你可以添加按钮控件，向按钮添加操作，然后运行“Hello World”应用查看效果。

向设计画布添加按钮

1. 在“解决方案资源管理器”中，双击“MainWindow.xaml”打开 XAML 标记编辑器。



可以在 XAML 编辑器中添加或更改标记。 与 UWP 项目不同，WinUI 3 没有“设计”视图。



2. 查看“窗口”根目录中“StackPanel”中嵌套的“按钮”控件。

```xaml
<Window
    x:Class="HelloWorld.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HelloWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button x:Name="myButton" Click="myButton_Click">Click Me</Button>
    </StackPanel>
</Window>
```

更改按钮上的标签

1. 在 XAML 编辑器中，将按钮内容的值从“Click me”更改为“Hello World!”。

```xaml
<Window
    x:Class="HelloWorld.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:HelloWorld"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button x:Name="myButton" Click="myButton_Click">Hello World!</Button>
    </StackPanel>
</Window>
```

2. 请注意，该按钮还指定了一个名为 myButton_Click 的 Click 事件处理程序。 我们将在下一步中讲到。

```xaml
<StackPanel Orientation="Horizontal" HorizontalAlignment="Center" VerticalAlignment="Center">
    <Button x:Name="myButton" Click="myButton_Click">Hello World!</Button>
</StackPanel>
```

修改事件处理程序

“事件处理程序”听起来复杂，但只不过是事件发生时，被调用代码的另一种名称。 在本例中，事件处理程序添加了“Hello World!”按钮触发的操作。

1. 在“解决方案资源管理器”中，双击代码隐藏页面 MainWindow.xaml.cs。
2. 在打开的 C# 编辑器窗口中编辑事件处理程序代码。

   现在，有趣的事情发生了。 默认的事件处理程序如下所示：

```
0 references
private void myButton_Click(object sender, RoutedEventArgs e)
{
    myButton.Content = "Clicked";
}
```

   让我们更改它，使它如下所示：

```
0 references
private async void myButton_Click(object sender, RoutedEventArgs e)
{
    var welcomeDialog = new ContentDialog()
    {
        Title = "Hello from HelloWorld",
        Content = "Welcome to your first Windows App SDK app.",
        CloseButtonText = "Ok",
        XamlRoot = myButton.XamlRoot
    };
    await welcomeDialog.ShowAsync();
}
```

   以下是可用于复制和粘贴的代码：

C#

```
private async void myButton_Click(object sender, RoutedEventArgs e)
{
    var welcomeDialog = new ContentDialog()
    {
        Title = "Hello from HelloWorld",
        Content = "Welcome to your first Windows App SDK app.",
        CloseButtonText = "Ok",
        XamlRoot = myButton.XamlRoot
    };
    await welcomeDialog.ShowAsync();
}
```

我们刚才做了什么？

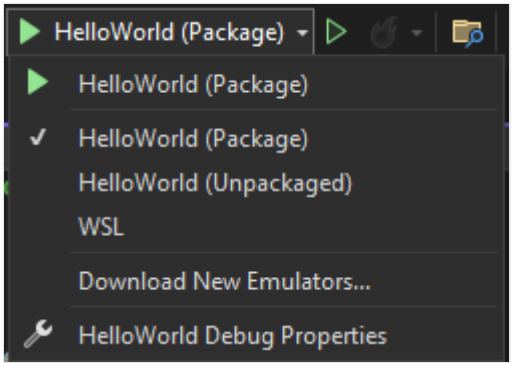
该代码使用 ContentDialog 控件在当前窗口中的模式弹出控件中显示欢迎消息。 （有关如何使用 `Microsoft.UI.Xaml.Controls.ContentDialog` 的详细信息，请参阅ContentDialog

类。）

运行应用程序

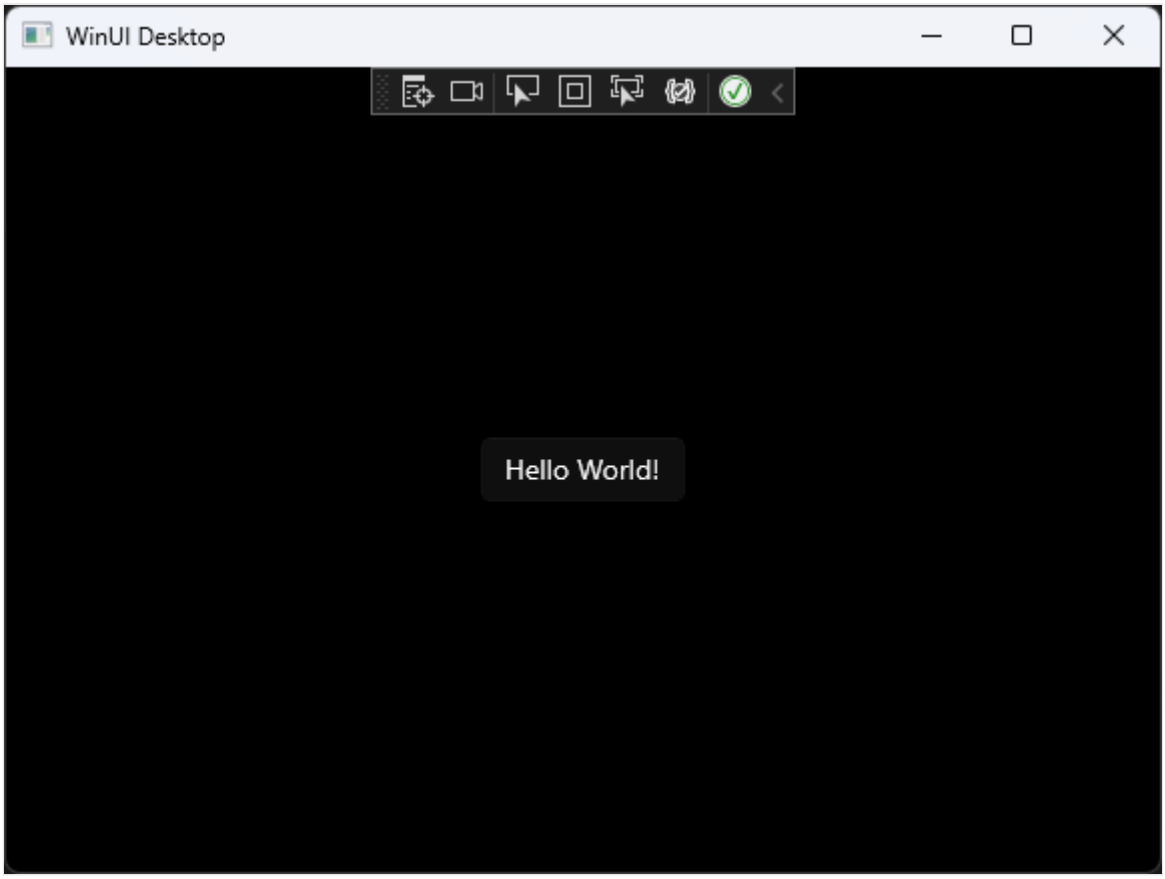
是时候生成、部署和启动“Hello World”Windows App SDK 应用，看看它是什么样子了。操作方法如下。

1. 使用“播放”按钮（它包含文本“HelloWorld (包)”）在本地计算机上启动应用程序。



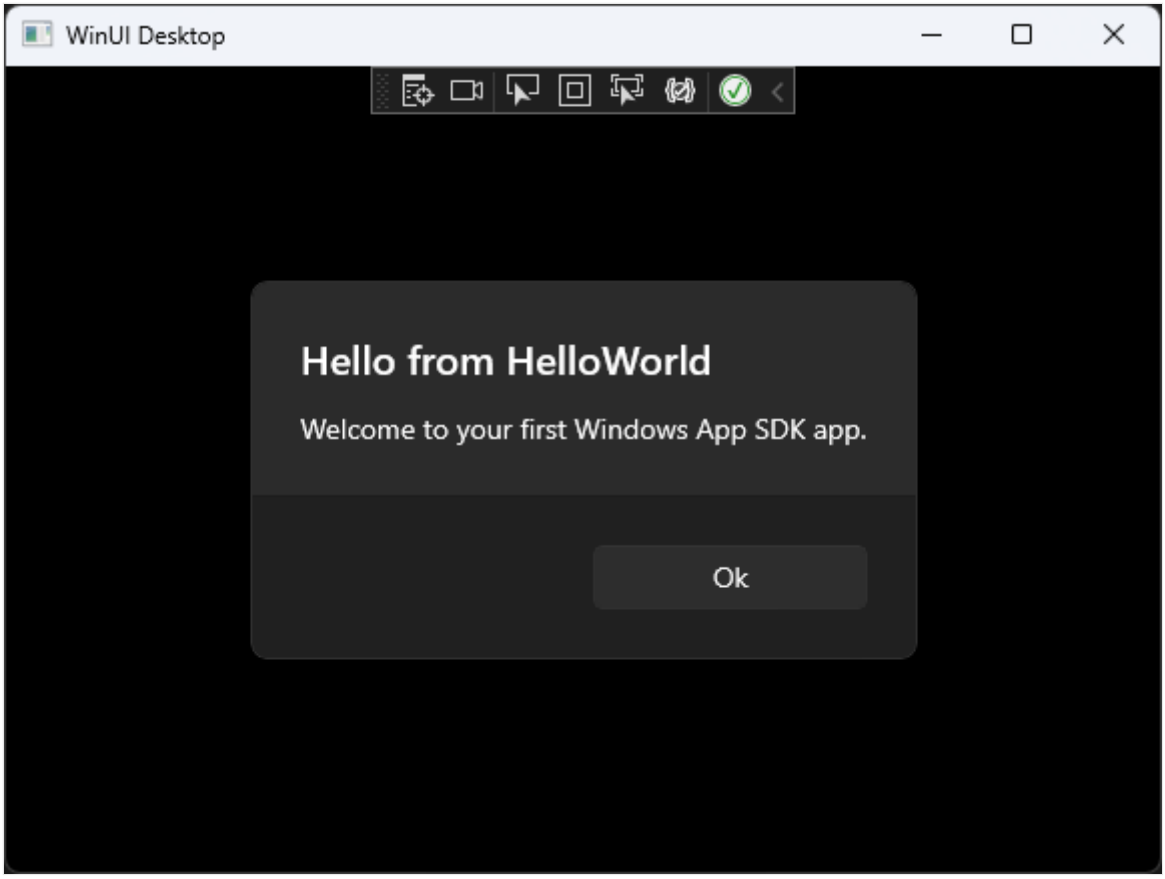
（或者，也可以在菜单栏选择“调试”>“开始调试”或按 F5 启动应用。）

2. 应用在初始屏幕消失后不久出现，请查看该应用。 此应用应类似于下图：



3. 选择“Hello world”按钮。

Windows 10 或更高版本的设备将显示一条消息，指出“欢迎使用第一个 Windows App SDK 应用”，标题为“Hello from HelloWorld”。单击“确定”关闭消息。



4. 要关闭应用，请在工具栏中选择“停止调试”按钮。 （或者，从菜单栏中选择“调试”>“停止调试”或按 Shift+F5。 ）

后续步骤

恭喜你完成本教程！ 我们希望你能了解一些与 Windows App SDK、WinUI 3 和 Visual Studio IDE 有关的基础知识。 要更加深入地了解，请继续学习下面的教程：

教程：使用 WinUI 3 创建简单的照片查看器

另请参阅

- 编写适用于 Windows 的应用：选择开发技术
- Windows App SDK 概述
- Windows App SDK/WinUI 3 示例

教程：在 Visual Studio 中使用 XAML 和 C# 创建第一个通用 Windows 平台应用程序

项目 • 2023/06/05

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

> **备注**
>
> 如果对通用 Windows 平台 (UWP) 中的当前功能感到满意，则无需将项目类型迁移到 Windows App SDK。 WinUI 2.x 和 Windows SDK 支持 UWP 项目类型。 如果要开始使用 WinUI 3 和 Windows App SDK，可以按照 **Windows App SDK 教程**中的步骤操作。

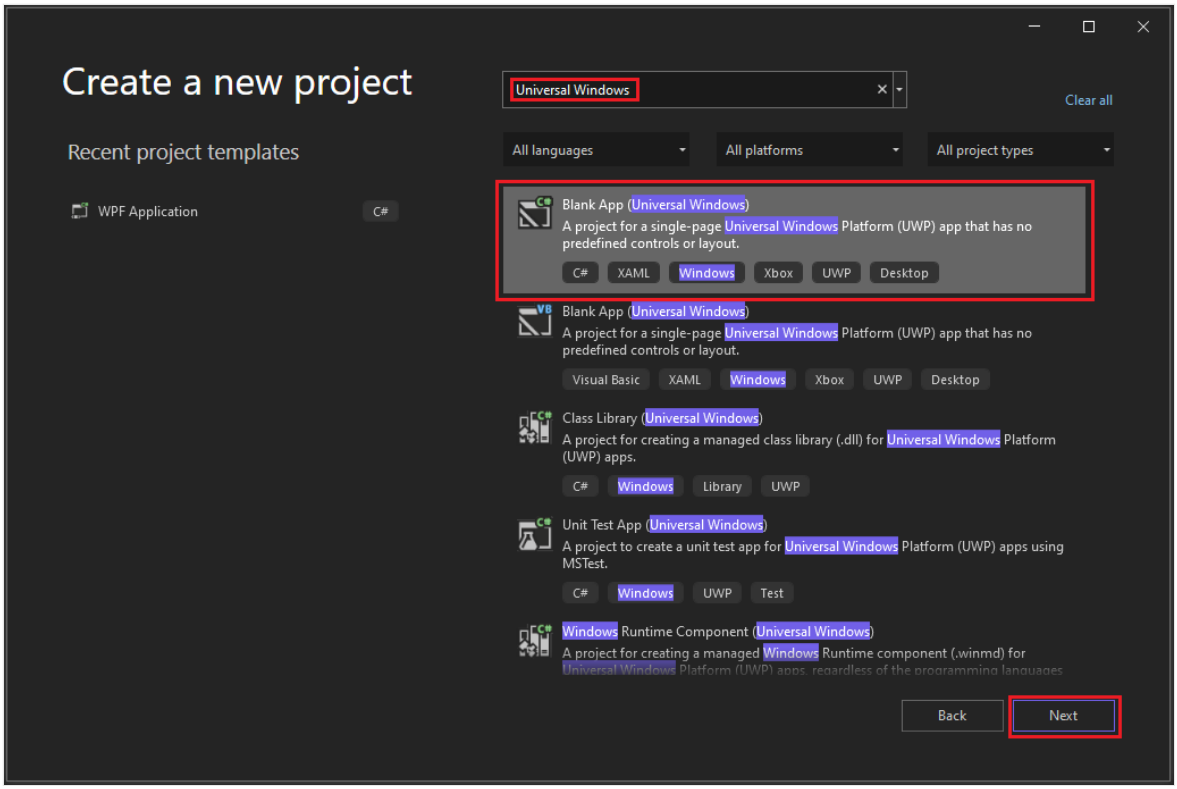
在这个 Visual Studio 集成开发环境 (IDE) 简介中，你将创建能在任何 Windows 10 或更高版本设备上运行的“Hello World”应用。 为此，将使用通用 Windows 平台 (UWP) 项目模板、Extensible Application Markup Language (XAML) 和 C# 编程语言。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

创建项目

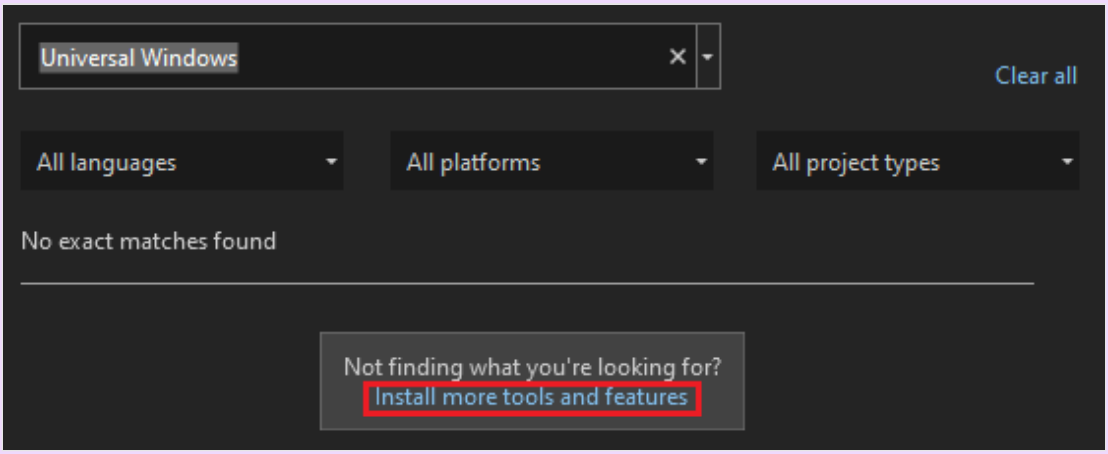
首先，创建通用 Windows 平台项目。 项目类型随附了所需的全部模板文件，无需添加任何内容！

1. 打开 Visual Studio，然后在“启动”窗口上，选择“创建新项目”。
2. 在“创建新项目”屏幕上，在搜索框中输入“通用 Windows”，选择“空白应用(通用 Windows)”对应的 C# 模板，然后选择“下一步”。

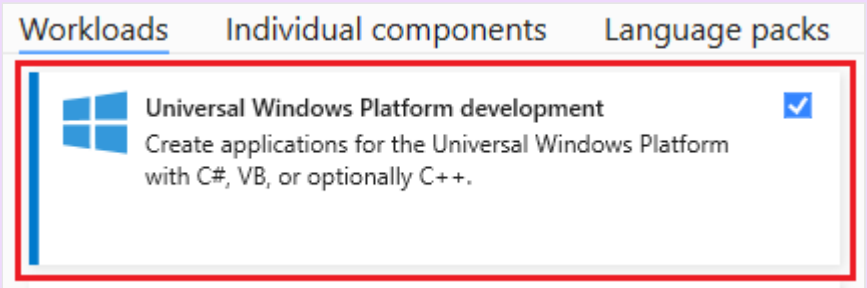


ⓘ 备注

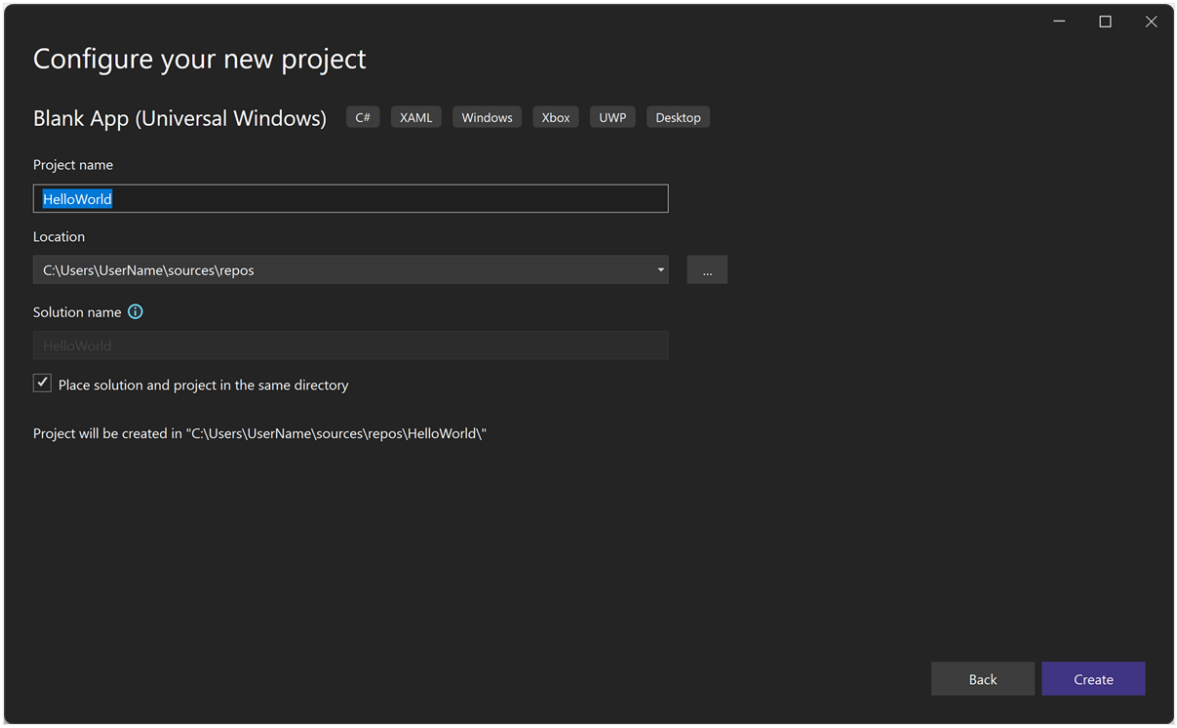
如果没有看到“空白应用(通用 Windows)” 项目模板，请单击“安装多个工具和功能” 链接。



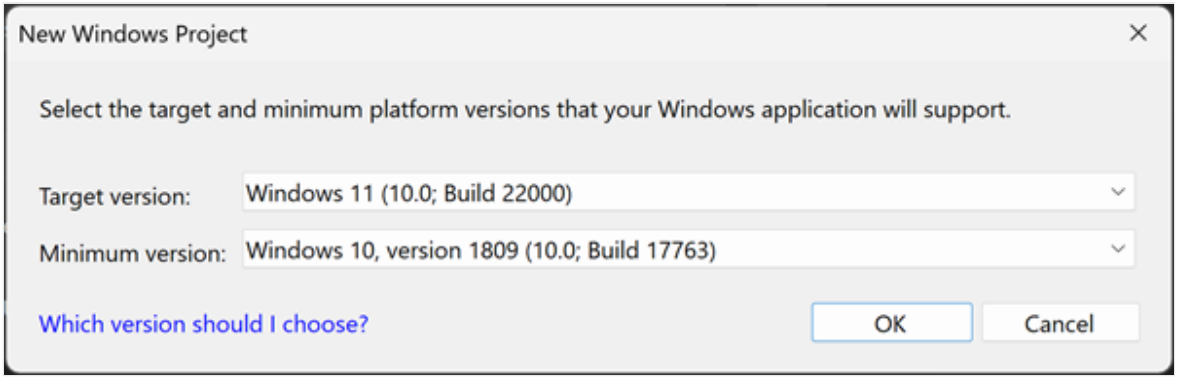
Visual Studio 安装程序启动。 选择“通用 Windows 平台开发”工作负载，然后选择“修改” 。



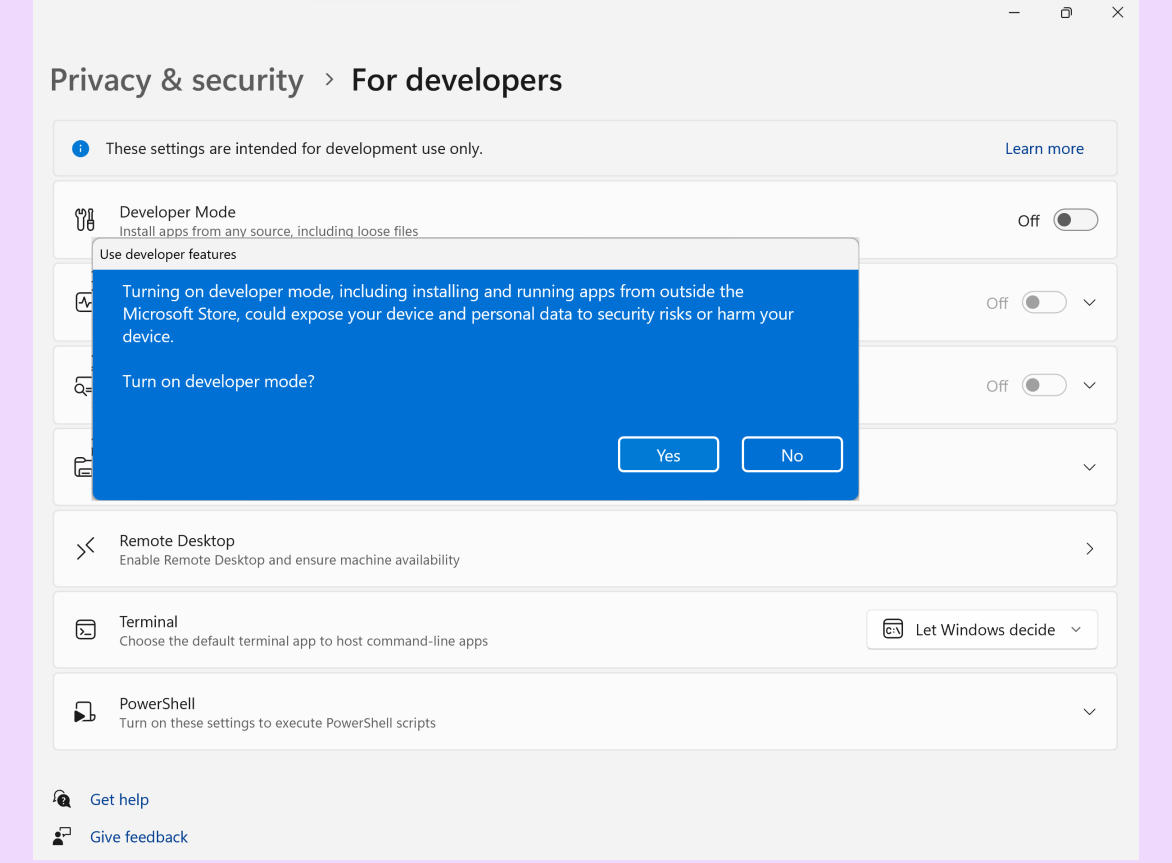
3. 为项目提供名称“HelloWorld”，然后选择“创建”。



4. 接接受“新式通用 Windows 平台项目”对话框中的默认目标版本和最小版本设置受“新式通用 Windows 平台项目” 对话框中的默认目标版本 和最小版本 设置。



> **备注**
>
> 如果这是你第一次使用 Visual Studio 创建 UWP 应用，将显示“启用 Windows 开发人员模式”对话框。 选择“开发人员设置”以打开“设置”。 打开“开发人员模式”，然后选择“是”。



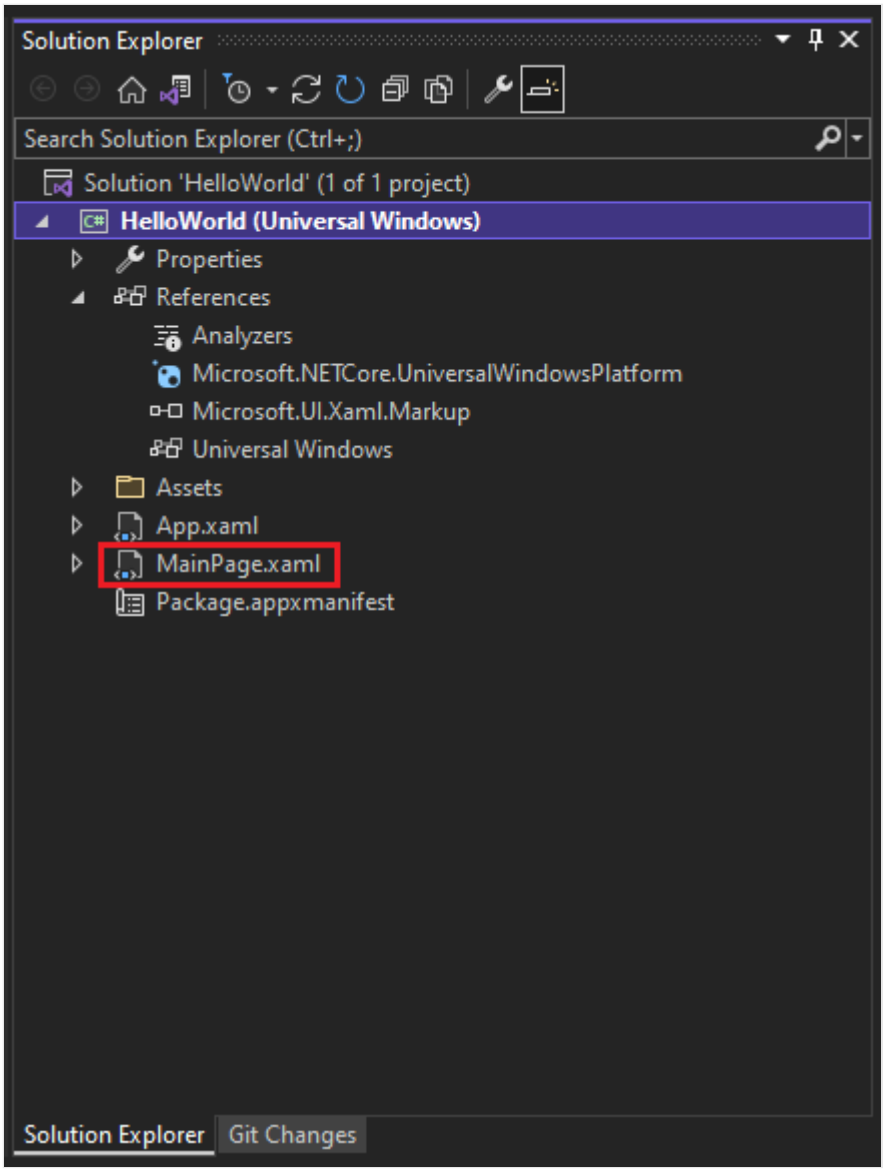
Visual Studio 会为你安装其他开发人员模式包。 包安装完成时，请关闭“设置”对话框 。

创建应用程序

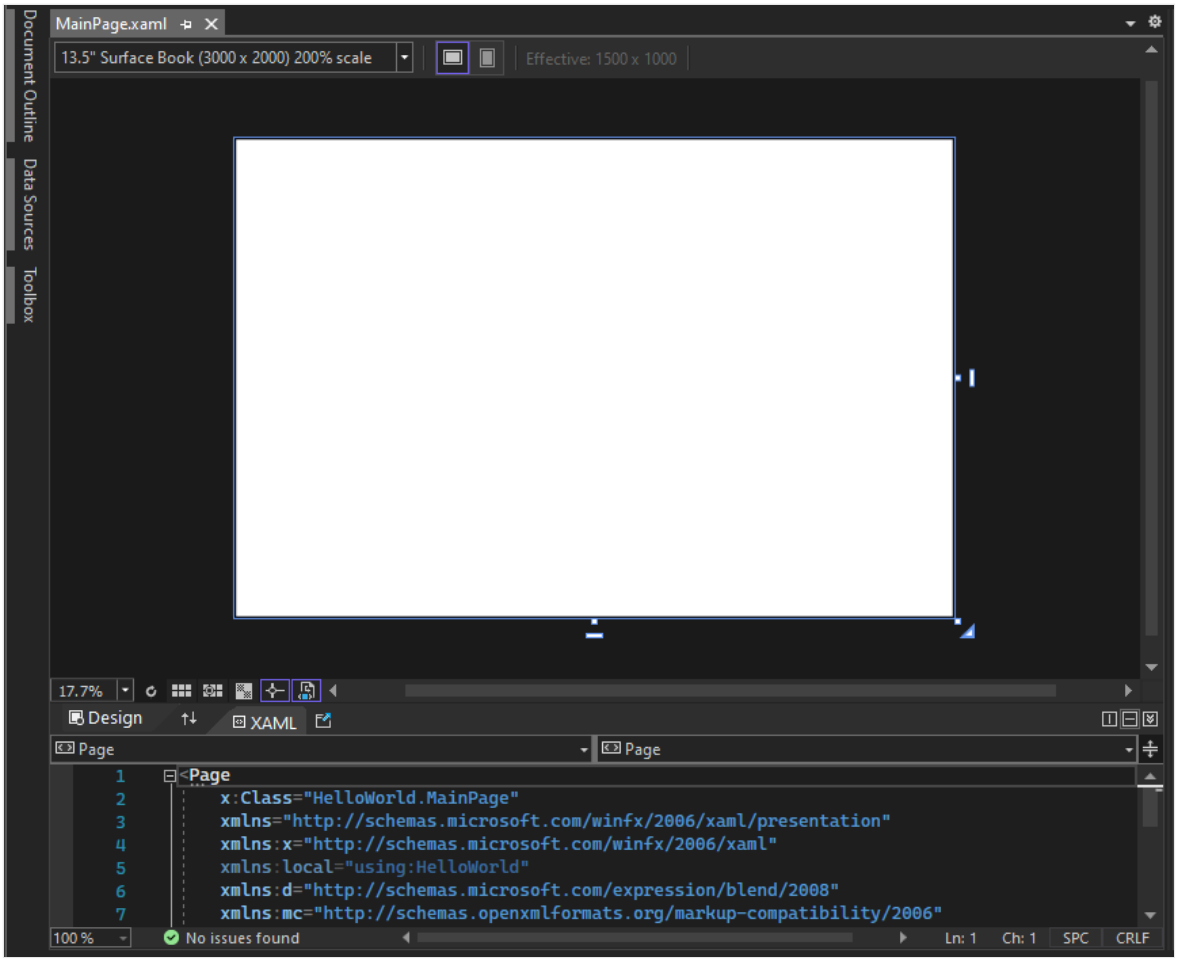
现在开始开发吧。 你可以添加按钮控件，向按钮添加操作，然后启用"Hello World"应用查看效果。

向设计画布添加按钮

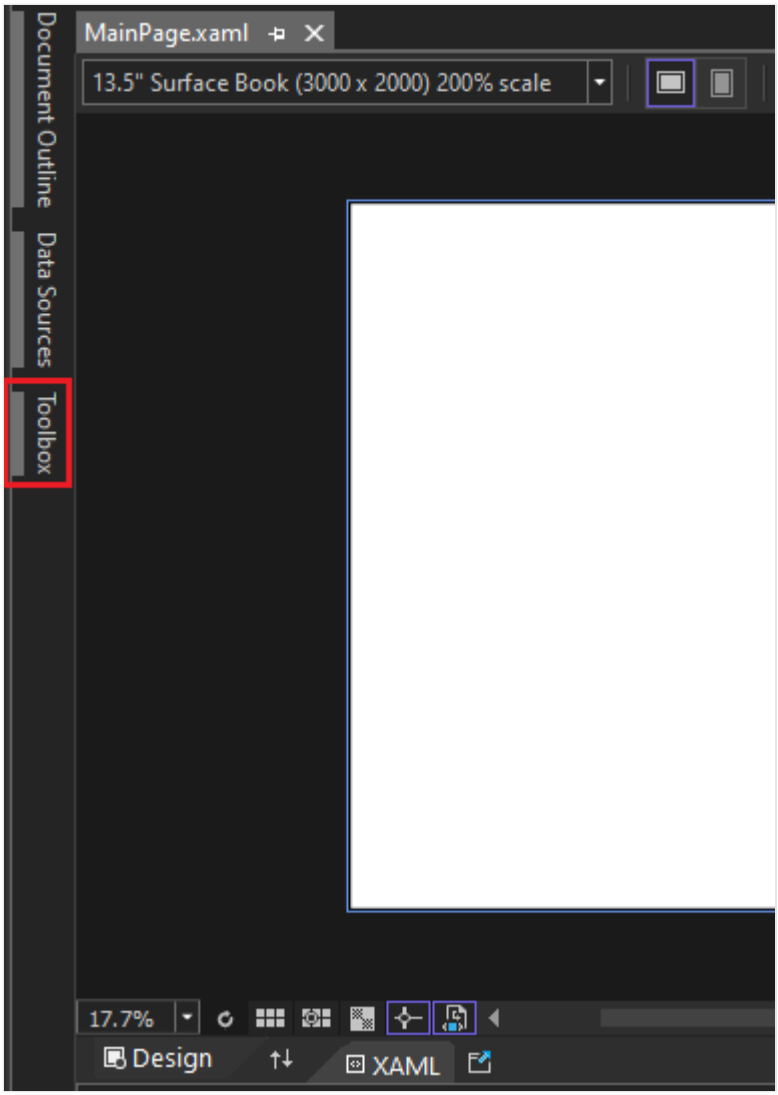
1. 在解决方案资源管理器中双击“MainPage.xaml”，打开拆分视图 。



出现两个窗格：一个是“XAML 设计器”，其中包含设计画布；另一个是“XAML 编辑器”，可用于添加或更改代码 。

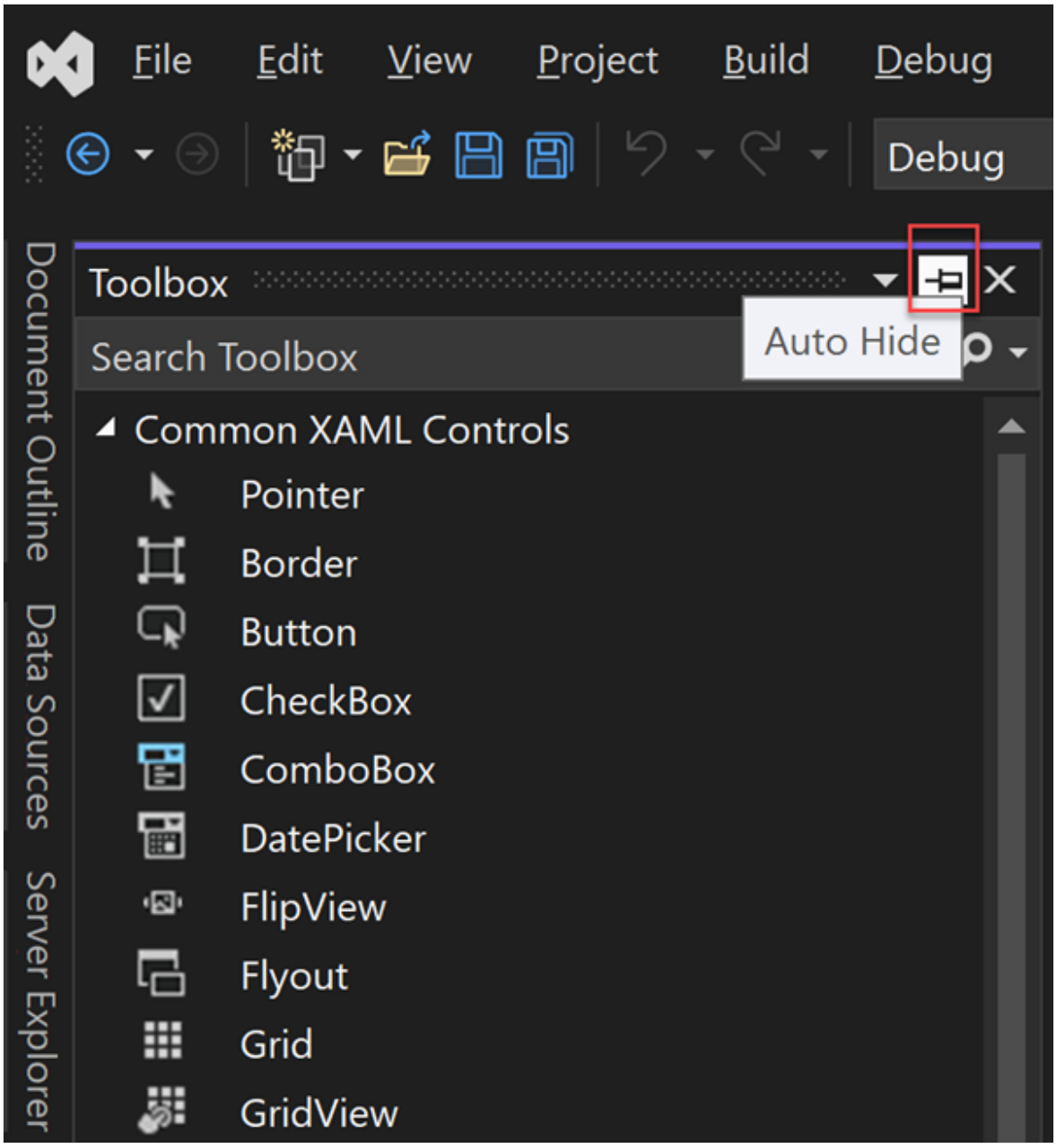


2. 选择“工具箱”，打开“工具箱”弹出窗口 。

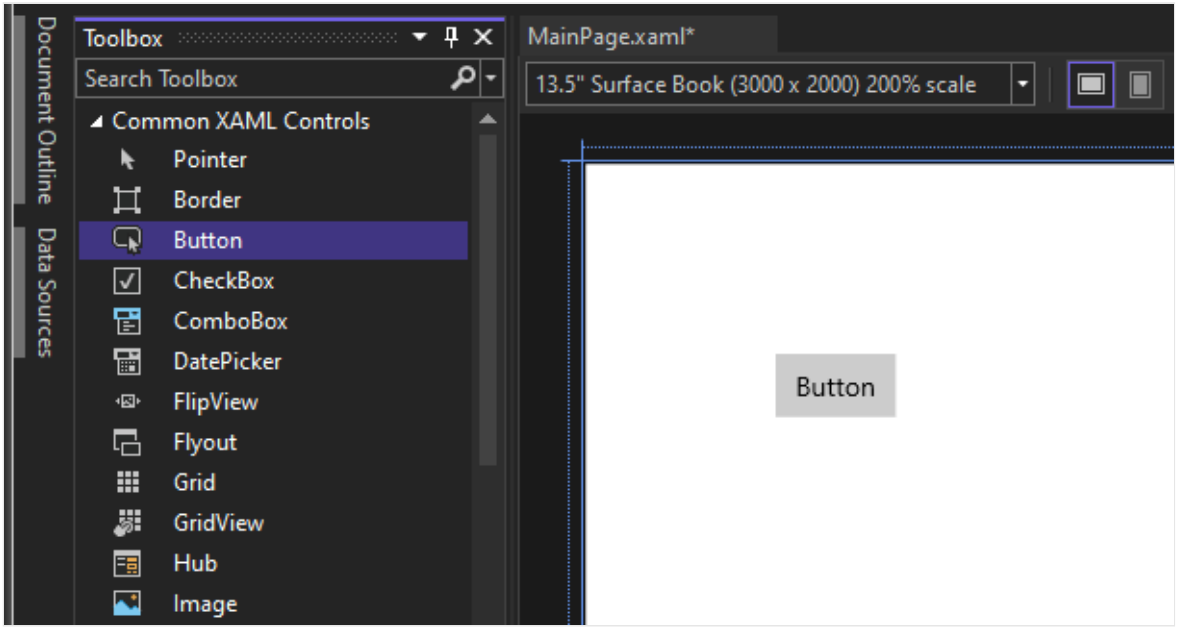


（如果没有看到“工具箱”选项，可以从菜单栏中打开它。为此，请选择“查看”>“工具栏”。或者，按 Ctrl+Alt+X。）

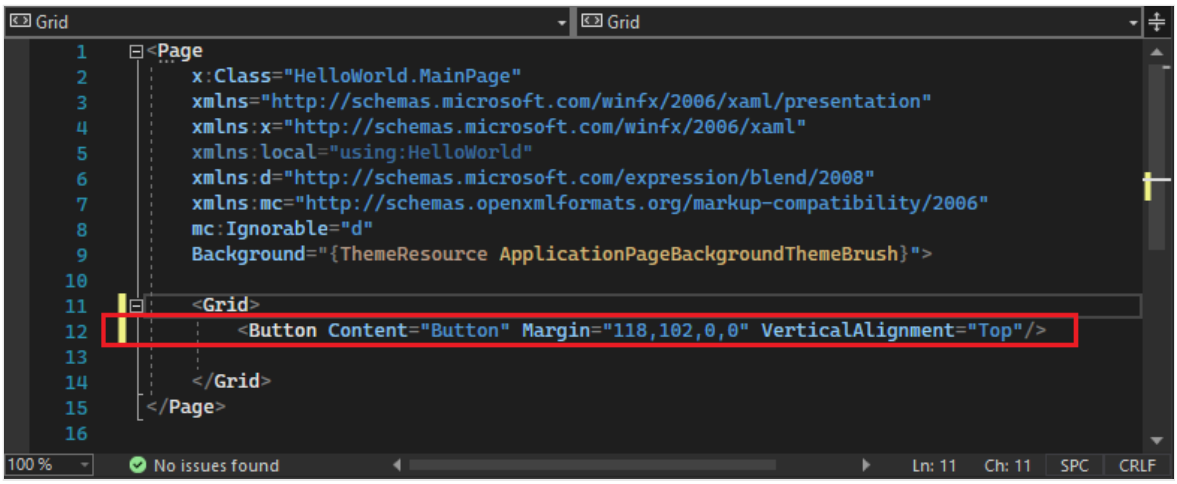
3. 选择“固定”图标，固定“工具箱”窗口。



4. 选择“按钮”控件，然后将其拖到设计画布上。



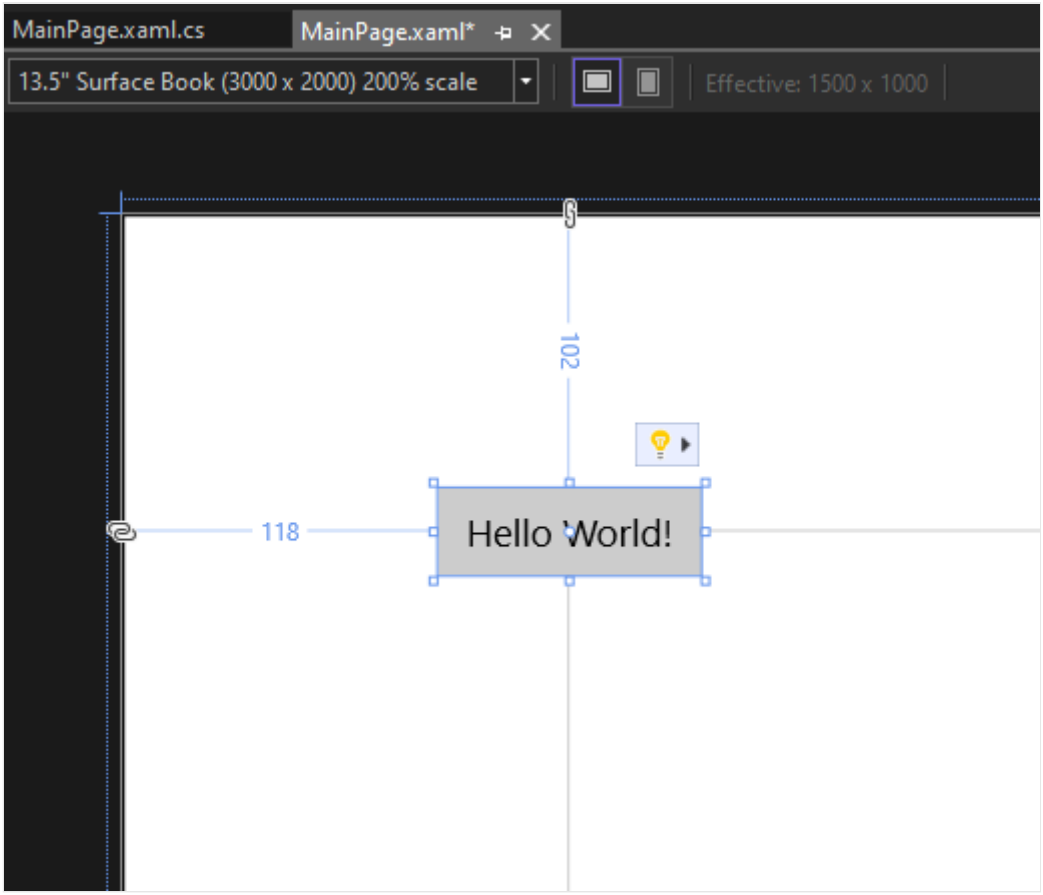
如果在 XAML 编辑器中查看代码，就会看到该按钮也已添加到此处：



向按钮添加标签

1. 在 XAML 编辑器中，将按钮内容的值从“按钮”更改为"Hello World!"。

2. 请注意 XAML 设计器中的按钮也会随之更改 。



添加事件处理程序

“事件处理程序”听起来复杂，但只不过是事件发生时，被调用代码的另一种名称。 在本例中，事件处理程序会将操作添加到“Hello World!”按钮。

1. 双击设计画布上的按钮控件。

2. 在 MainPage.xaml.cs（代码隐藏页）中编辑事件处理程序代码 。

   现在，有趣的事情发生了。 默认事件处理程序如下所示：

```
30          private void Button_Click(object sender, RoutedEventArgs e)
31          {
32
33          }
```

   让我们更改它，使它如下所示：

```
23      public sealed partial class MainPage : Page
24      {
            0 references
25          public MainPage()
26          {
27              this.InitializeComponent();
28          }
29
            0 references
30          private async void Button_Click(object sender, RoutedEventArgs e)
31          {
32              MediaElement mediaElement = new MediaElement();
33              var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
34              Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream = await
35                  synth.SynthesizeTextToStreamAsync("Hello, World!");
36              mediaElement.SetSource(stream, stream.ContentType);
37              mediaElement.Play();
38          }
39      }
40  }
41
```

   以下是可用于复制和粘贴的代码：

C#

```
private async void Button_Click(object sender, RoutedEventArgs e)
{
   MediaElement mediaElement = new MediaElement();
   var synth = new Windows.Media.SpeechSynthesis.SpeechSynthesizer();
   Windows.Media.SpeechSynthesis.SpeechSynthesisStream stream = await
synth.SynthesizeTextToStreamAsync("Hello, World!");
   mediaElement.SetSource(stream, stream.ContentType);
   mediaElement.Play();
}
```

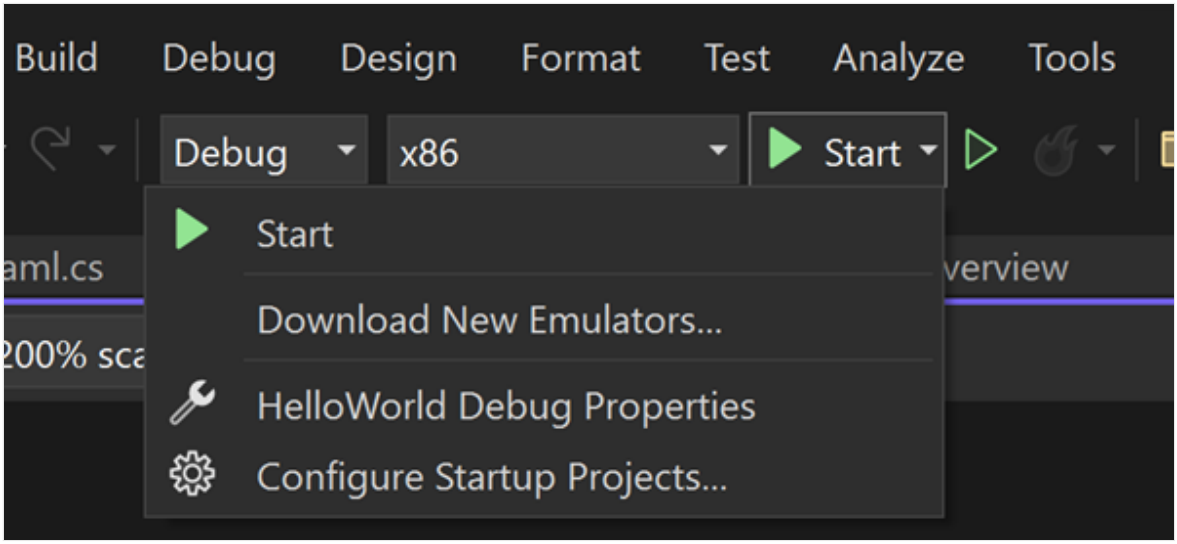
我们刚才做了什么？

代码使用某些 Windows API 创建语音合成对象，然后向此对象通过一些文本用于朗读。（有关使用 `SpeechSynthesis` 的详细信息，请参阅 System.Speech.Synthesis。）

运行此应用程序

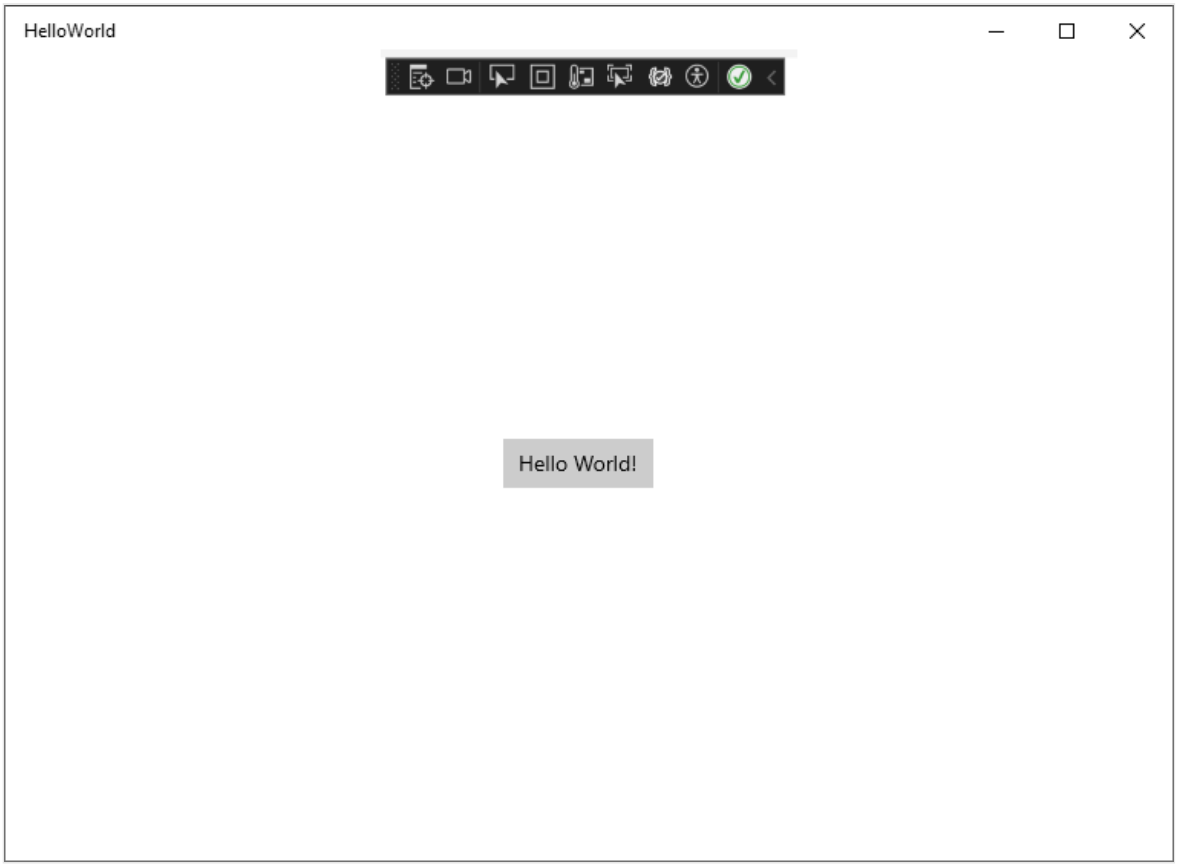
该生成、部署和启动“Hello World”UWP 应用了，以了解它的视听效果。 操作方法如下。

1. 使用“播放”按钮（它包含文本“本地计算机”）在本地计算机上启动应用程序。



（或者，也可以在菜单栏选择“调试”>“开始调试”或按 F5 启动应用。）

2. 应用在初始屏幕消失后不久出现，请查看该应用。 此应用应类似于下图：



3. 选择“Hello world”按钮。

   Windows 10 或更高版本设备将逐字说“Hello, World!”。

4. 要关闭应用，请在工具栏中选择“停止调试”按钮。 （或者，从菜单栏中选择“调试”>“停止调试”或按 Shift+F5。 ）

后续步骤

恭喜你完成本教程！ 我们希望你能了解一些与 UWP 和 Visual Studio IDE 有关的基础知识。 要更加深入地了解，请继续学习下面的教程：

创建用户界面

请参阅

- UWP 概述
- 获取 UWP 应用示例

教程：使用 C# 创建简单 WPF 应用

项目 • 2023/11/17

通过完成本教程，你将熟悉在使用 Visual Studio 开发应用程序时可使用的许多工具、对话框和设计器。 你将创建“Hello, World”应用程序、设计 UI、添加代码并调试错误。在此期间，你将了解如何使用集成开发环境 (IDE)。

先决条件

- 如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。
- 确保安装了 .NET 桌面开发工作负载。 可以在 Visual Studio 安装程序中验证此配置。
- 在本教程中，可以使用 .NET Framework 或 .NET Core。 .NET Core 是较新的、更新式的框架。 .NET Core 需要 Visual Studio 2019 16.3 或更高版本。

什么是 WPF?

Windows Presentation Foundation (WPF) 是一个可创建桌面客户端应用程序的 UI（用户界面）框架。 WPF 开发平台支持广泛的应用开发功能，包括应用模型、资源、控件、图形、布局、数据绑定、文档和安全性。

WPF 是 .NET 的一部分，因此，如果你曾经使用 ASP.NET 或 Windows 窗体通过 .NET 构建应用程序，应该会熟悉此编程体验。 WPF 使用 Extensible Application Markup Language (XAML) 为应用程序编程提供声明性模型。 有关详细信息，请参阅 WPF .NET 概述。

配置 IDE

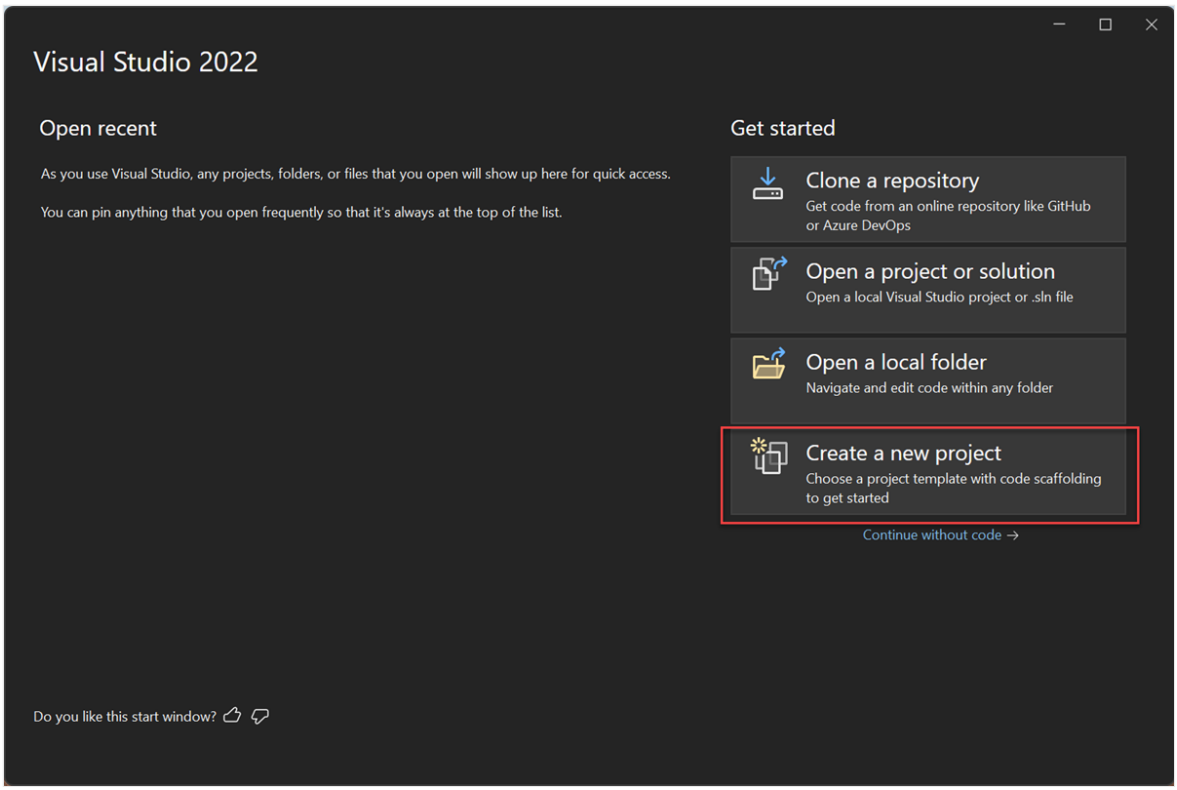
启动 Visual Studio 时，“启动”窗口首先打开。 选择“继续但无需代码”打开开发环境。 将看到工具窗口、菜单和工具栏，以及主窗口空间。 工具窗口位于应用程序窗口的左右两侧。 搜索框、菜单栏和标准工具栏位于顶部。 加载解决方案或项目时，编辑器和设计器显示在应用程序窗口中间。 开发应用程序时，大部分时间都在此中心区域。

创建项目

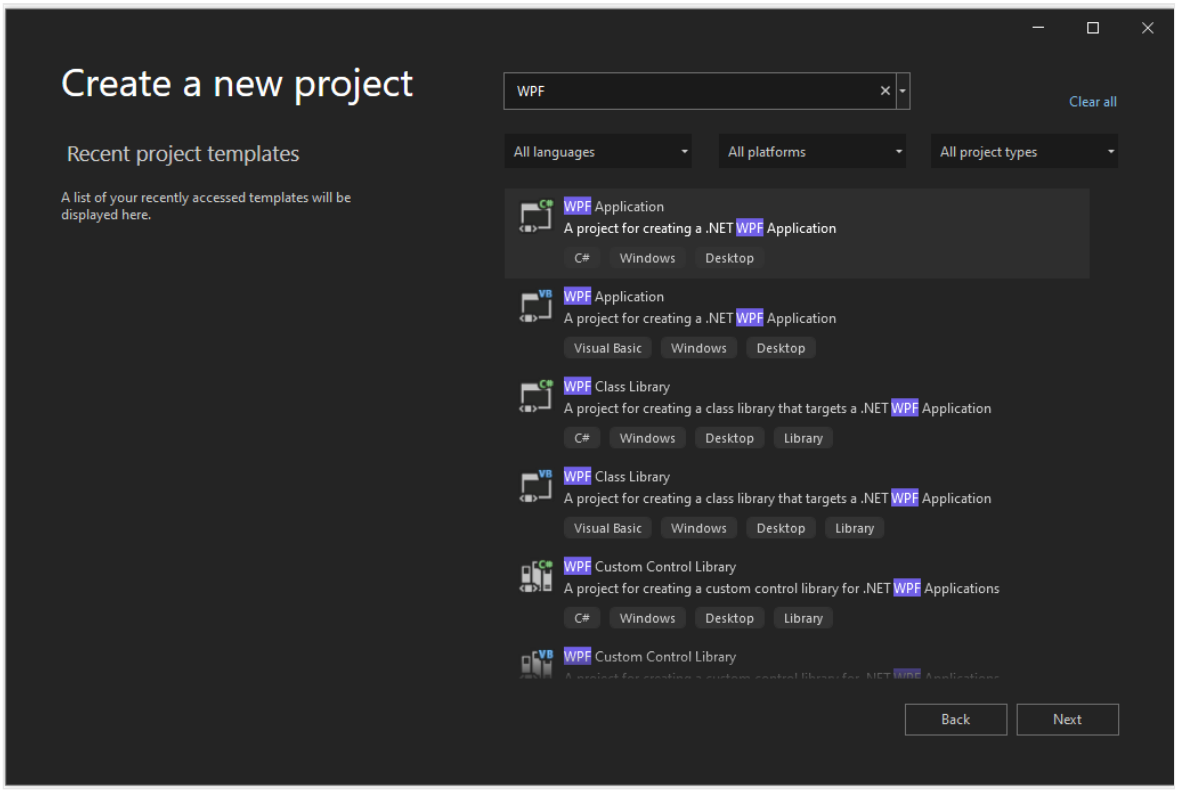
在 Visual Studio 中创建应用程序时，应首先创建项目和解决方案。 在本示例中，你将创建一个 Windows Presentation Foundation (WPF) 项目。

1. 打开 Visual Studio。

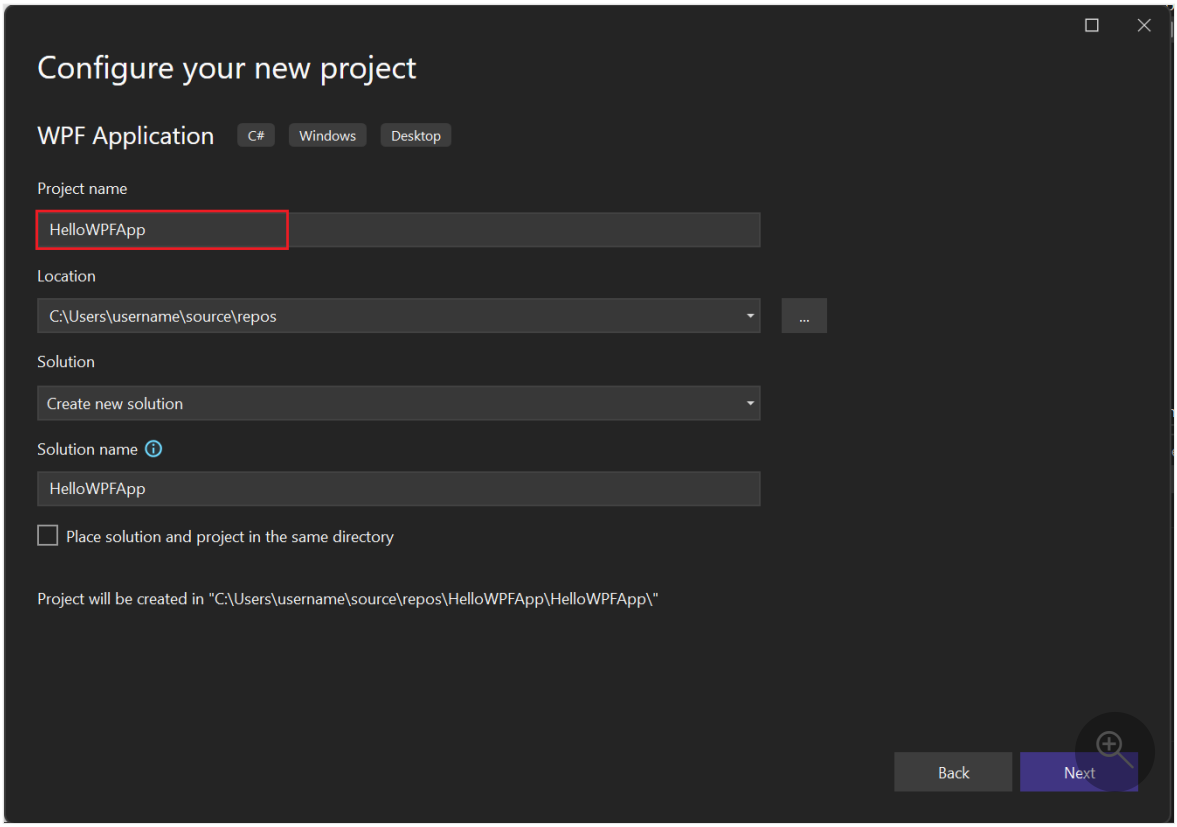
2. 在“开始”窗口上，选择“创建新项目”。



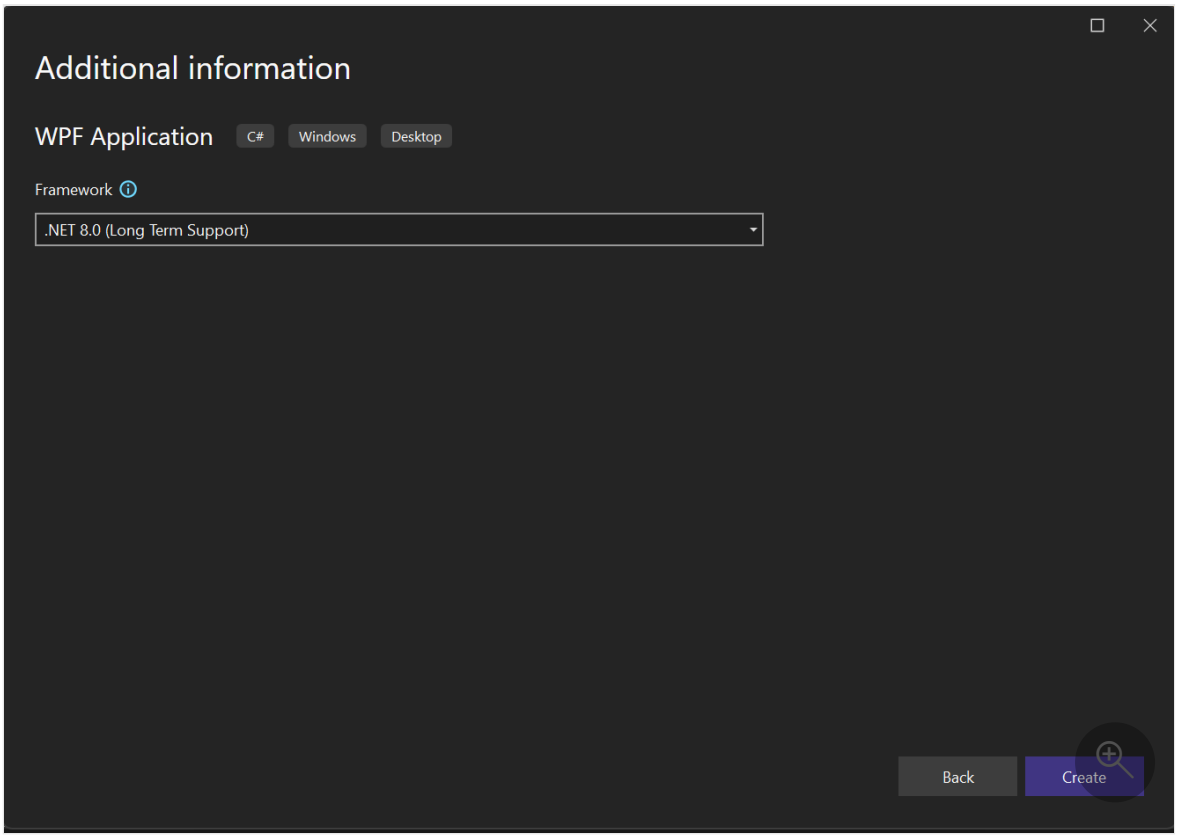
3. 在“创建新项目”屏幕上，搜索“WPF”，选择“WPF 应用程序”，然后选择“下一步”。



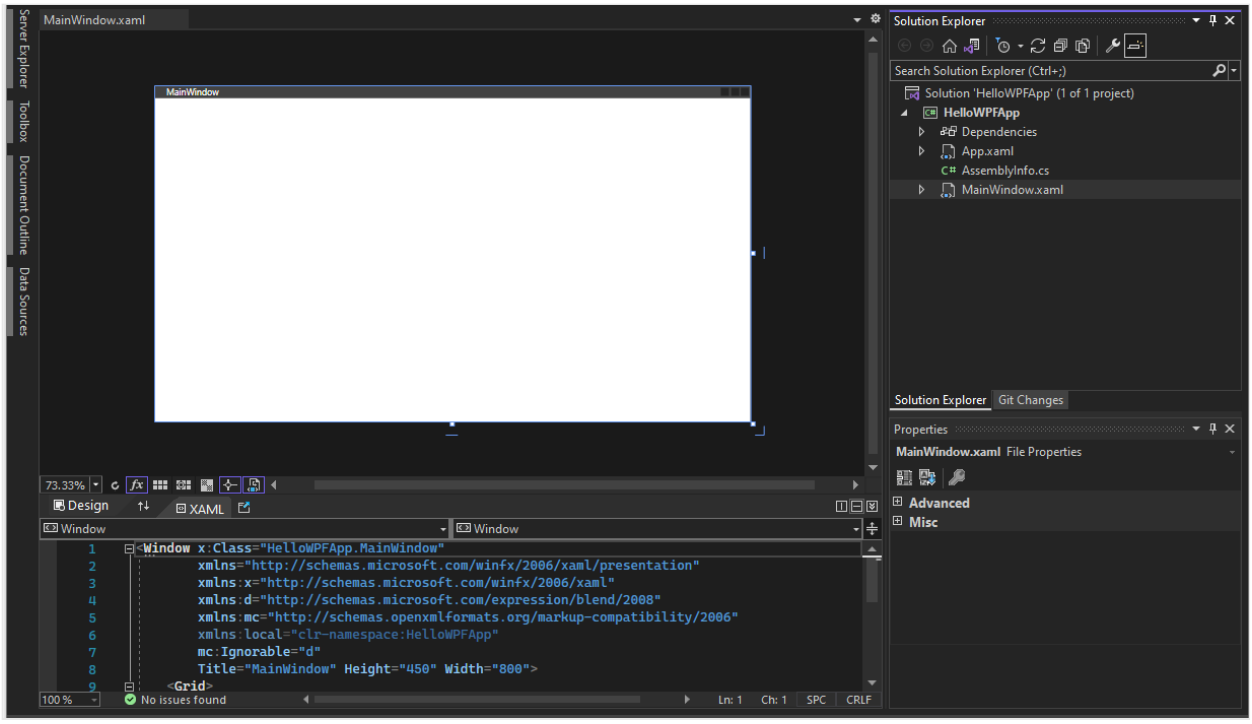
4. 在下一个屏幕中，为项目指定名称“HelloWPFApp”，然后选择“下一步”。



5. 在“其他信息”窗口中，验证是否为目标框架选择了“.NET 8.0”。 然后，选择“创建”。

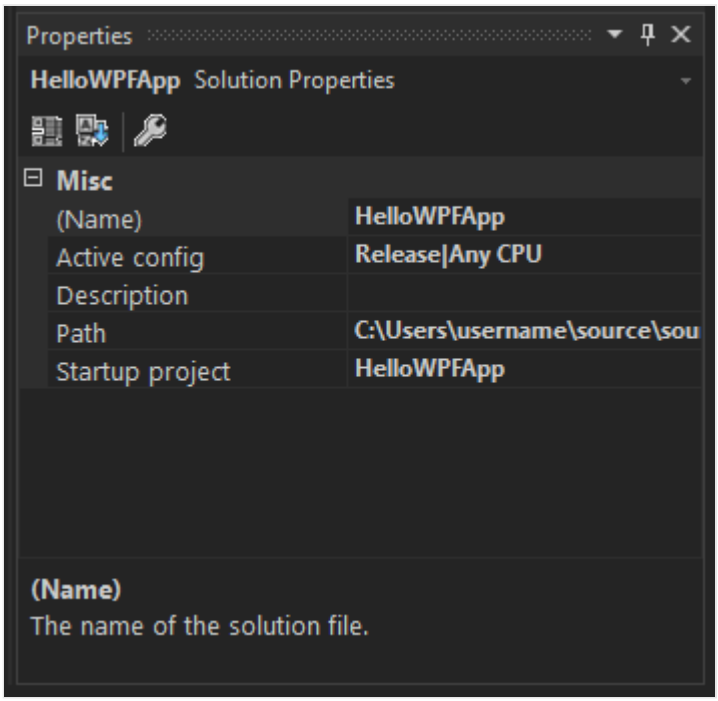


Visual Studio 将创建 HelloWPFApp 项目和解决方案，“解决方案资源管理器”将显示各种文件。 “WPF 设计器”在拆分视图中显示 MainWindow.xaml 的设计视图和 XAML 视图。 您可以滑动拆分器，以显示任一视图的更多或更少部分。 您可以选择只查看可视化视图或 XAML 视图。



> **① 备注**
>
> 若要详细了解 XAML (eXtensible Application Markup Language)，请参阅 **WPF 的 XAML 概述**页。

你可以在创建项目后进行自定义。 若要执行此操作，请从“视图”菜单中选择“属性窗口”或按 F4 。 然后可显示和更改应用程序中的项目项、控件和其他项的选项。



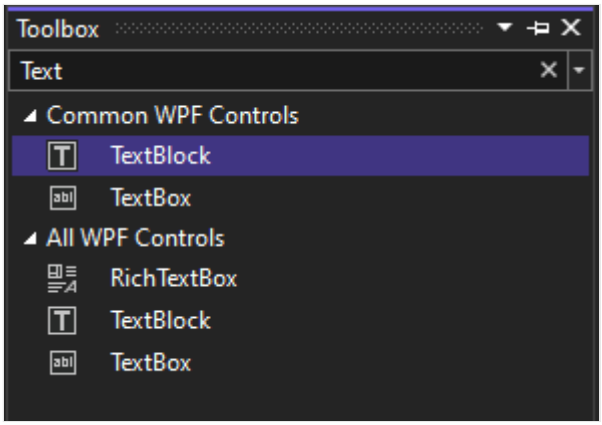
设计用户界面 (UI)

如果设计器未打开，请选择“MainWindow.xaml”，然后按 Shift+F7 打开设计器。

我们会将三种类型的控件添加到此应用程序：一个 TextBlock 控件、两个 RadioButton 控件和一个 Button 控件。

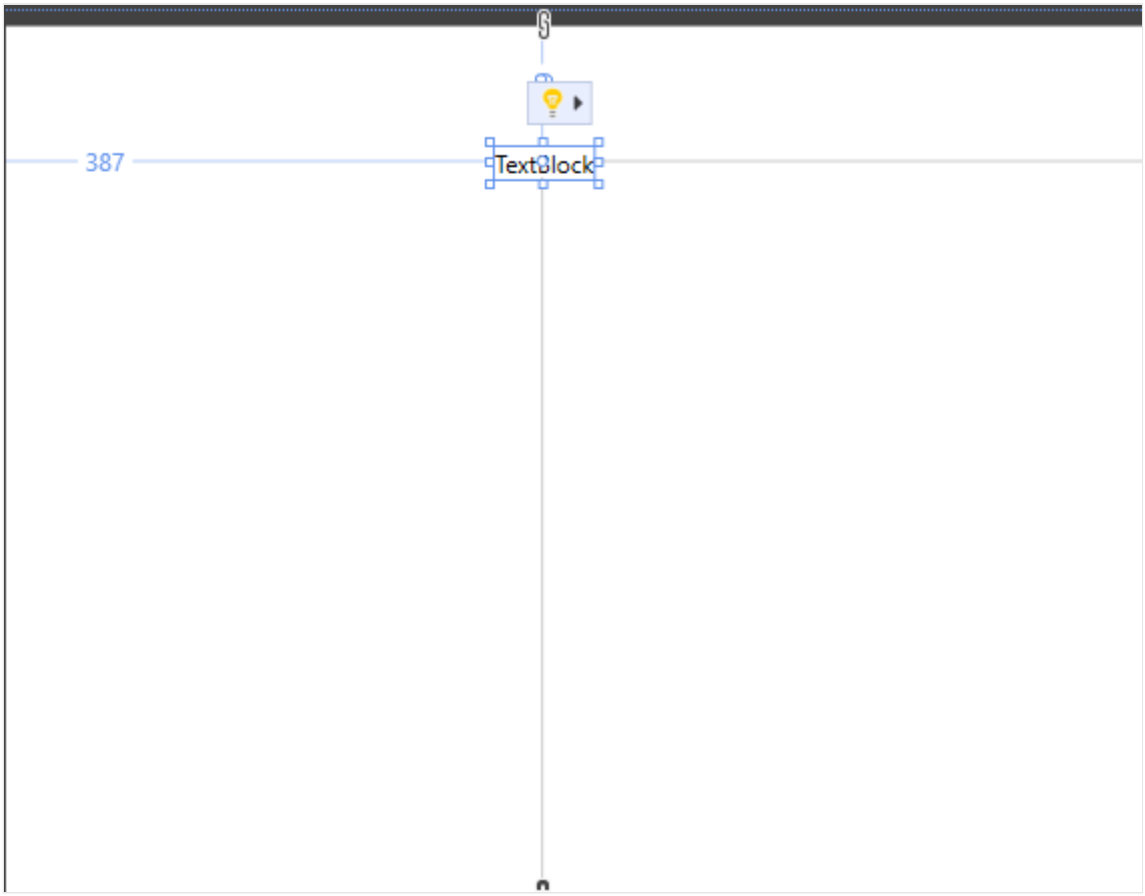
添加 TextBlock 控件

1. 按“Ctrl+Q”激活搜索框，然后键入“工具箱” 。 从结果列表中选择“查看”>“工具箱”。
2. 在“工具箱”中，展开“公共 WPF 控件”节点以查看 TextBlock 控件。



3. 通过选择“TextBlock”项并将其拖到设计图面的窗口中，将 TextBlock 控件添加到设计图面中。 把控件居中到窗口的顶部附近。 可以使用参考线将控件居中。

   你的窗口应类似于下图：



XAML 标记应如下面的示例所示：

```
XAML

<Grid>
    <TextBlock HorizontalAlignment="Left" Margin="387,60,0,0"
TextWrapping="Wrap" Text="TextBlock" VerticalAlignment="Top"/>
</Grid>
```

自定义文本块中的文本

1. 在 XAML 视图中，找到 TextBlock 的标记并将 Text 属性从 `TextBox` 更改为 `Select a message option and then choose the Display button.`

   XAML 标记应如下面的示例所示：

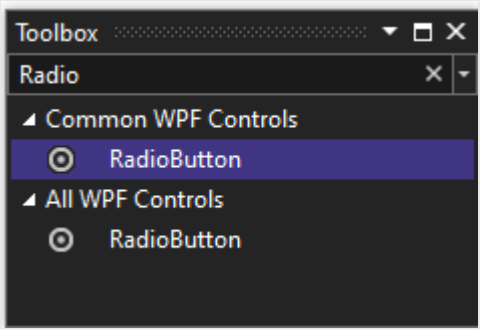
   ```
   XAML

   <Grid>
       <TextBlock HorizontalAlignment="Left" Margin="387,60,0,0"
   TextWrapping="Wrap" Text="Select a message option and then choose the
   Display button." VerticalAlignment="Top"/>
   </Grid>
   ```

2. 如果愿意，再次使 TextBlock 居中，然后通过按 Ctrl+S 或使用“文件” 菜单项保存更改。

接下来，向窗体添加两个 RadioButton 控件。

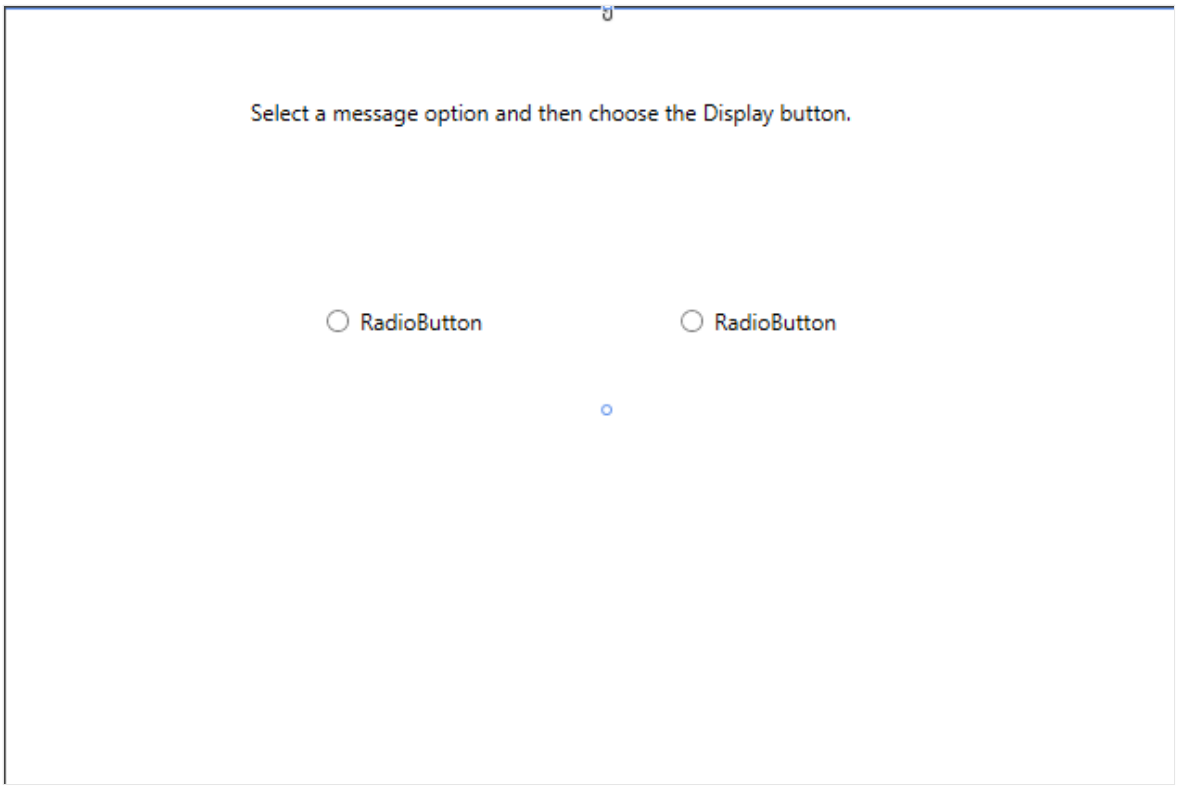
添加单选按钮

1. 在“工具箱”中，查找“RadioButton”控件。

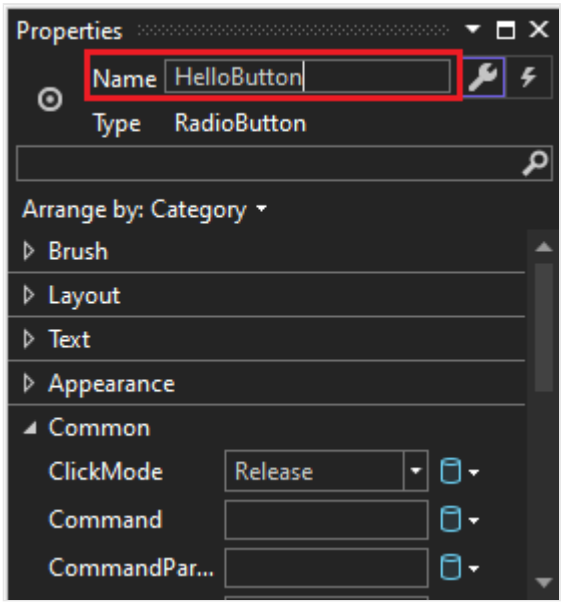
   

2. 通过选择“RadioButton”项并将其拖到设计图面的窗口中，将两个 RadioButton 控件添加到设计图面中。 移动按钮（通过选择它们并使用箭头键），以便按钮并排显示在 TextBlock 控件下。 可以使用参考线来对齐控件。

你的窗口应如下所示：



3. 在左侧 RadioButton 控件的 “**属性**” 窗口中，将 “**名称**” 属性（位于 “**属性**” 窗口顶部）更改为 `HelloButton`。



4. 在右侧 RadioButton 控件的“属性”窗口中，将“名称”属性更改为 `GoodbyeButton`，然后保存更改。

接下来，为每个 RadioButton 控件添加显示文本。 以下程序将更新 RadioButton 控件的 “**内容**” 属性。

添加每个单选按钮的显示文本

1. 在 XAML 中将两个单选按钮 `HelloButton` 和 `GoodbyeButton` 的“**内容**”属性更新为 `"Hello"` 和 `"Goodbye"`。 XAML 标记现在应类似于以下示例：

```
XAML

<Grid>
     <TextBlock HorizontalAlignment="Left" Margin="252,47,0,0"
TextWrapping="Wrap" Text="Select a message option and then choose the
Display button." VerticalAlignment="Top"/>
     <RadioButton x:Name="HelloButton" Content="Hello"
HorizontalAlignment="Left" Margin="297,161,0,0"
VerticalAlignment="Top"/>
     <RadioButton x:Name="GoodbyeButton" Content="Goodbye"
HorizontalAlignment="Left" Margin="488,161,0,0"
VerticalAlignment="Top"/>
</Grid>
```

设置要默认选中的单选按钮

在这一步中，我们将 HelloButton 设置为默认选中，这样两个单选按钮中始终有一个处于选中状态。

1. 在 XAML 视图中，找到 HelloButton 的标记。
2. 添加 IsChecked 属性，并将其设置为“True”。 具体而言，添加 `IsChecked="True"`。

   XAML 标记现在应类似于以下示例：

```
XAML

<Grid>
     <TextBlock HorizontalAlignment="Left" Margin="252,47,0,0"
TextWrapping="Wrap" Text="Select a message option and then choose the
Display button." VerticalAlignment="Top"/>
     <RadioButton x:Name="HelloButton" Content="Hello" IsChecked="True"
HorizontalAlignment="Left" Margin="297,161,0,0"
VerticalAlignment="Top"/>
     <RadioButton x:Name="GoodbyeButton" Content="Goodbye"
HorizontalAlignment="Left" Margin="488,161,0,0"
VerticalAlignment="Top"/>
</Grid>
```

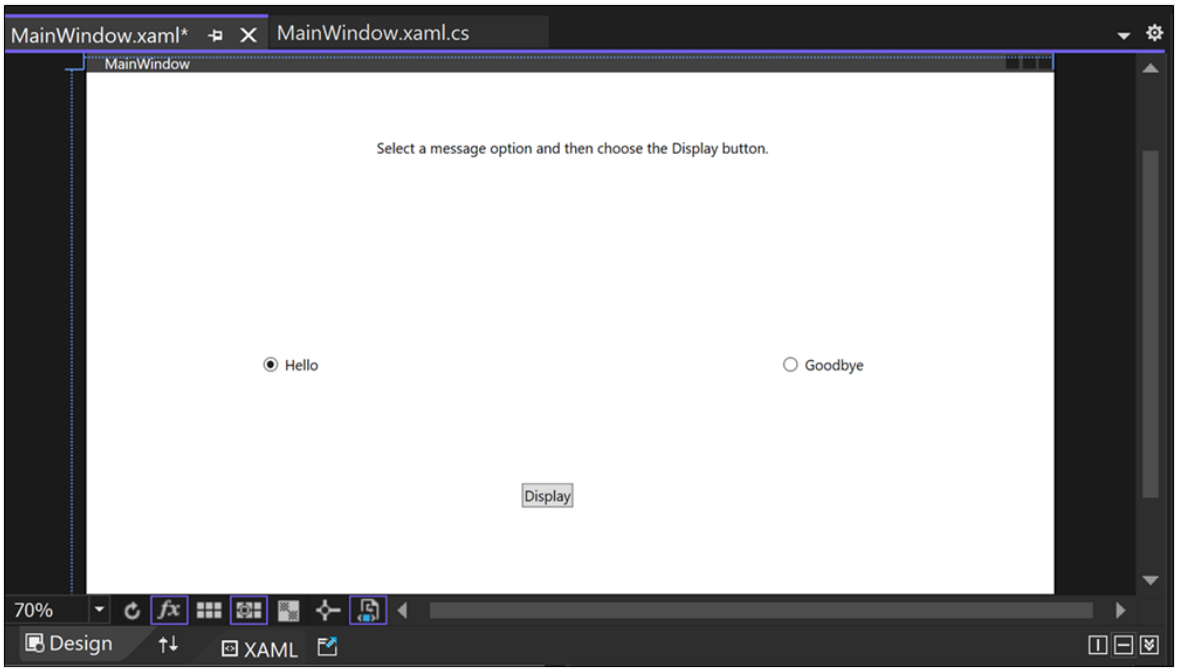
最后添加的 UI 元素是一个 Button 控件。

添加按钮控件

1. 在“工具箱”中，找到“按钮”控件，然后通过将控件拖到设计视图的窗体中，将其添加到 RadioButton 控件下方的设计界面中。 这些参考线可以帮助你将控件居中。

2. 在 XAML 视图中，将 Button 控件的“内容”值从 `Content="Button"` 更改为 `Content="Display"`，然后保存更改。

   你的窗口应类似于以下屏幕截图。



   XAML 标记现在应类似于以下示例：

XAML

```
<Grid>
     <TextBlock HorizontalAlignment="Left" Margin="252,47,0,0"
TextWrapping="Wrap" Text="Select a message option and then choose the
Display button." VerticalAlignment="Top"/>
     <RadioButton x:Name="HelloButton" Content="Hello" IsChecked="True"
HorizontalAlignment="Left" Margin="297,161,0,0"
VerticalAlignment="Top"/>
     <RadioButton x:Name="GoodbyeButton" Content="Goodbye"
HorizontalAlignment="Left" Margin="488,161,0,0"
VerticalAlignment="Top"/>
     <Button Content="Display" HorizontalAlignment="Left"
Margin="377,270,0,0" VerticalAlignment="Top" Width="75"/>
</Grid>
```

向显示按钮添加代码

此应用程序运行时，用户选择单选按钮，再选择“显示” 按钮之后，会显示一个消息框。 选择 Hello 将显示一个消息框，选择 Goodbye 将显示另一个。 若要创建此行为，请将代码添加到 MainWindow.xaml.cs 中的 `Button_Click` 事件。

1. 在设计图面上，双击 “**显示**” 按钮。

   此时，MainWindow.xaml.cs 打开，光标位于 `Button_Click` 事件上。

```C#
private void Button_Click(object sender, RoutedEventArgs e)
{

}
```

   双击“显示”按钮时，`Click="Button_Click"` 将添加到 XAML。

   XAML 标记现在应类似于以下示例：

```XAML
<Grid>
     <TextBlock HorizontalAlignment="Left" Margin="252,47,0,0"
TextWrapping="Wrap" Text="Select a message option and then choose the
Display button." VerticalAlignment="Top"/>
     <RadioButton x:Name="HelloButton" Content="Hello" IsChecked="True"
HorizontalAlignment="Left" Margin="297,161,0,0"
VerticalAlignment="Top"/>
     <RadioButton x:Name="GoodbyeButton" Content="Goodbye"
HorizontalAlignment="Left" Margin="488,161,0,0"
VerticalAlignment="Top"/>
     <Button Content="Display" HorizontalAlignment="Left"
Margin="377,270,0,0" VerticalAlignment="Top" Width="75"
Click="Button_Click"/>
</Grid>
```

2. 输入以下代码：

```C#
if (HelloButton.IsChecked == true)
{
     MessageBox.Show("Hello.");
}
else if (GoodbyeButton.IsChecked == true)
{
    MessageBox.Show("Goodbye.");
}
```

3. 保存应用程序。

调试并测试应用程序

接下来将调试应用程序，查找错误并测试两个消息框是否正确显示。 下面的说明介绍如何生成和启动调试器，但以后可以阅读生成 WPF 应用程序 (WPF) 和调试 WPF 获取有关详细信息。

更改 MainWindow.xaml 的名称

为 MainWindow 指定更具体的名称。 在“解决方案资源管理器”中，右键单击“MainWindow.xaml”，然后选择“重命名”。 将该文件重命名为“Greetings.xaml”。

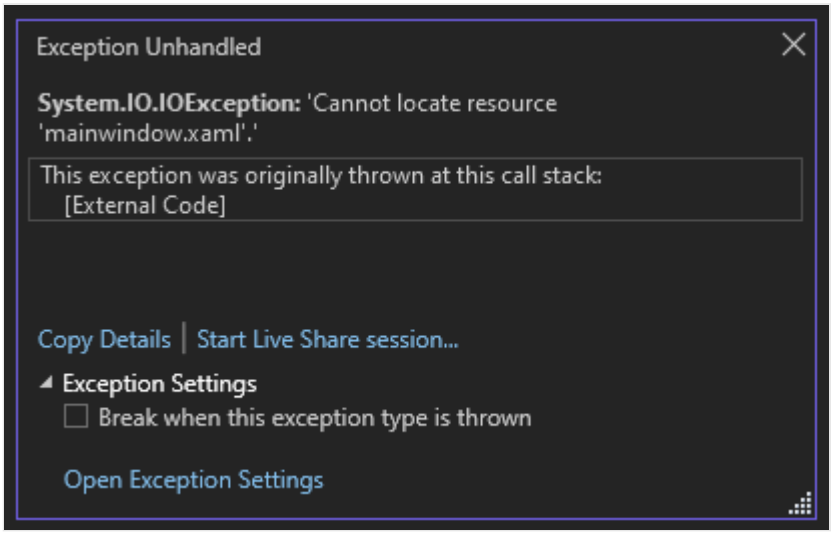
查找并修复错误

在此步骤中，将遇到之前因更改 MainWindow.xaml 文件的名称而引起的错误。

开始调试和查找错误

1. 通过按 F5或选择“调试”，然后选择“启动调试”，启动调试程序。

   此时将出现“中断模式”窗口，“输出”窗口指示“发生 IOException: 找不到资源 mainwindow.xaml”。

2. 依次选择“调试”>“停止调试”，停止调试程序。

开始学习本教程时，我们将 MainWindow.xaml 重命名为 Greetings.xaml，但是该代码仍然引用 MainWindow.xaml 作为应用程序的启动 URI，因此该项目无法启动。

将 Greetings.xaml 指定为启动 URI

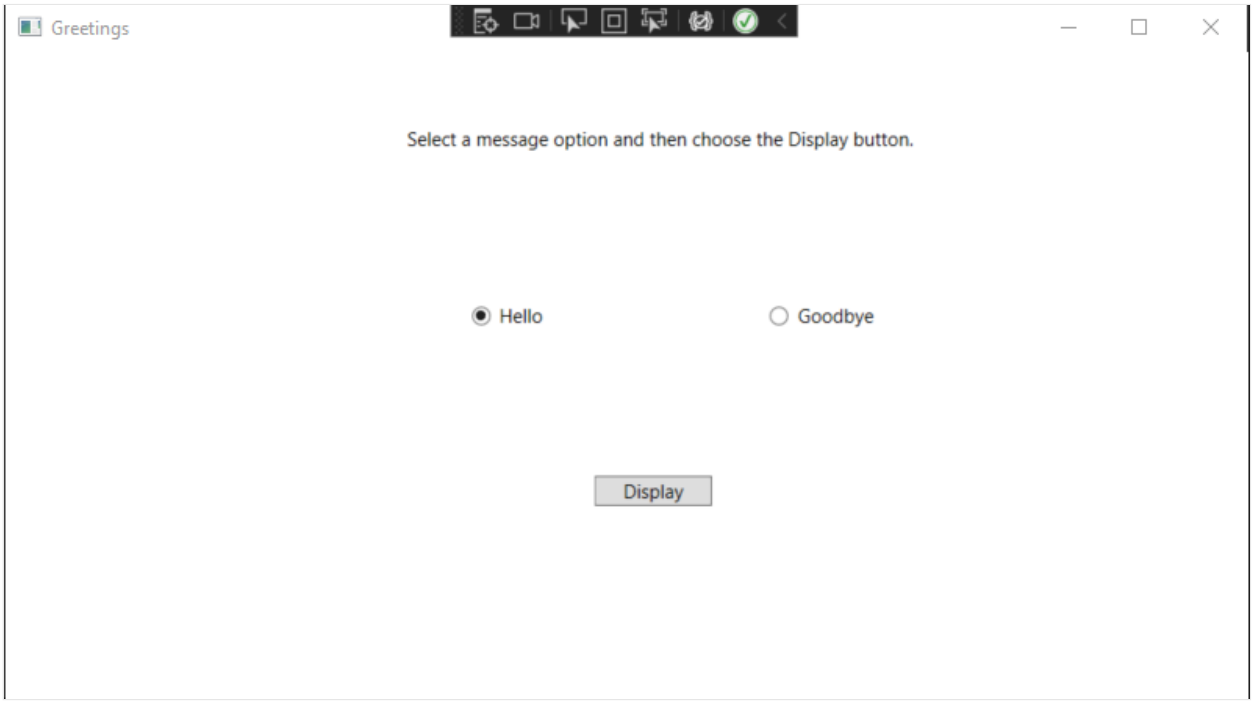
1. 在“解决方案资源管理器”中，打开“App.xaml”文件。

2. 将 `StartupUri="MainWindow.xaml"` 更改为 `StartupUri="Greetings.xaml"`，然后保存更改。

作为一个可选步骤，将应用程序窗口标题更改为匹配此新名称可以避免混淆。

1. 在“解决方案资源管理器”中，打开刚刚重命名的 Greetings.xaml 文件。
2. 将 Window.Title 属性的值从 `Title="MainWindow"` 更改为 `Title="Greetings"`，然后保存更改。

再次启动调试程序 （按“F5”）。 你现在应该可以看到应用程序的 Greetings 窗口。



现在关闭应用程序窗口，停止调试。

使用断点进行调试

可通过添加一些断点，在调试期间测试代码。 可以通过选择“调试”>“切换断点”、通过在编辑器中想要添加断点的代码行旁边的左边距中单击或按 F9 来添加断点。

添加断点

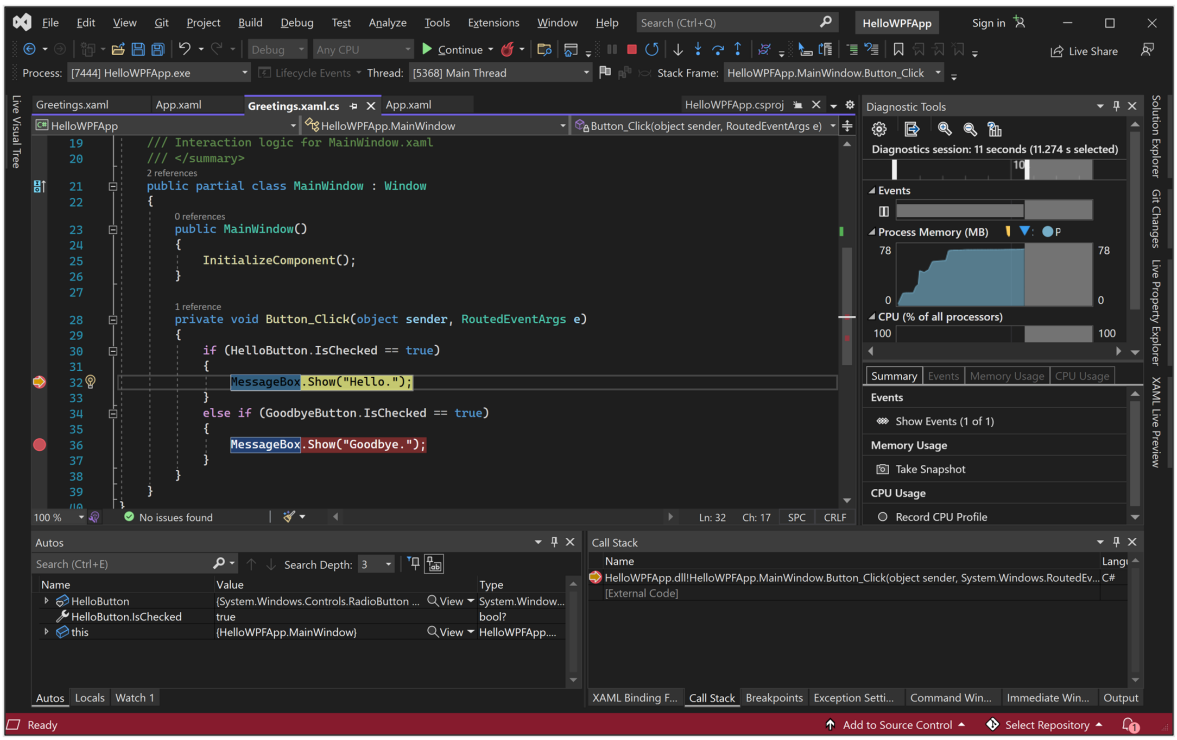
1. 打开“Greetings.xaml.cs”，并选择以下行：`MessageBox.Show("Hello.")`
2. 通过选择 “**调试**” -> “**切换断点**”，从菜单中添加断点。

   编辑器窗口最左侧边距中该代码行附近将显示一个红圈。
3. 选择以下行： `MessageBox.Show("Goodbye.")`。

4. 按“F9”键添加断点，然后按“F5”启动调试。

5. 在 “**Greetings**” 窗口中，选择 “**Hello**” 单选按钮，然后选择 “**显示**” 按钮。

   行 `MessageBox.Show("Hello.")` 将用黄色突出显示。 在 IDE 底部，“自动”、“本地”和“监视”窗口一起停靠在左侧，而“调用堆栈”、“断点”、“异常设置”、“命令”、“即时”和“输出”窗口一起停靠在右侧。



6. 在菜单栏上，选择“调试”>“跳出”。

   应用程序继续执行，并将显示出带有“Hello”的消息框。

7. 选择消息框上的 “**确定**” 按钮将其关闭。

8. 在 “**Greetings**” 窗口中，选择 “**Goodbye**” 单选按钮，然后选择 “**显示**” 按钮。

   行 `MessageBox.Show("Goodbye.")` 将用黄色突出显示。

9. 按“F5”键继续调试。 当消息框出现时，选择消息框上的 “**确定**” 按钮将其关闭。

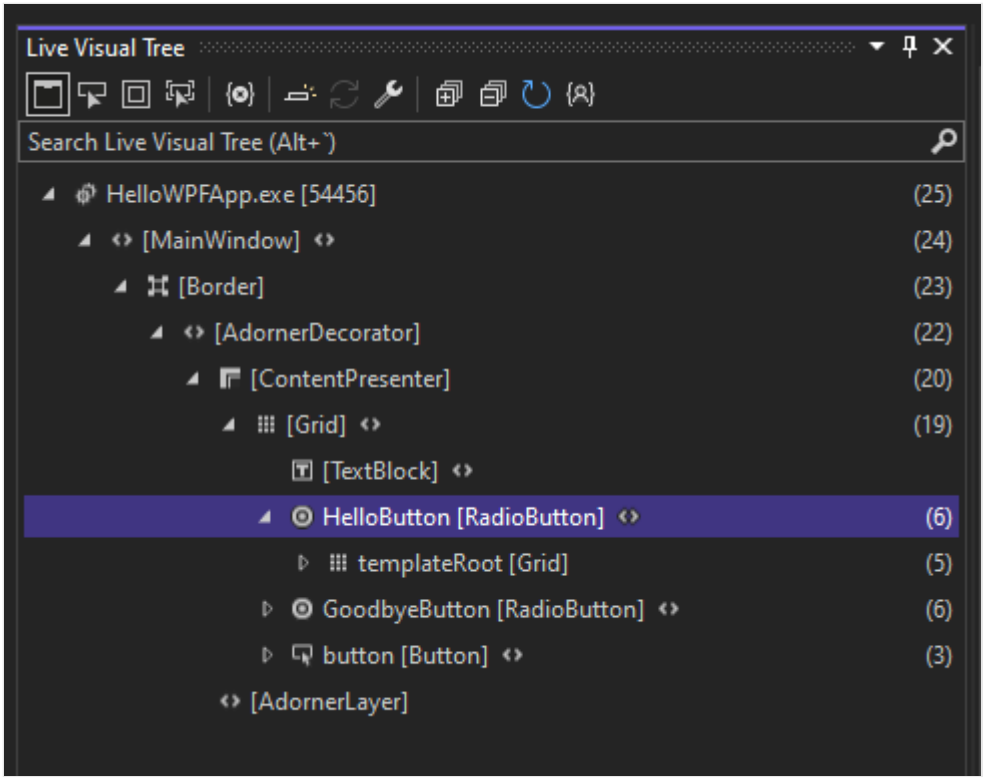
10. 关闭应用程序窗口，停止调试。

11. 在菜单栏上，选择“调试”>“禁用所有断点”。

查看 UI 元素的表示形式

在正在运行的应用中，你会看到窗口顶部显示了一个小组件。 该小组件是一个运行时帮助程序，通过它可以快速访问一些有用的调试功能。 选择第一个按钮“转到实时可视化

树”。 随即会看到一个包含一个树的窗口，树中包含页面的所有可视元素。 展开节点，找到你添加的按钮。



生成应用程序的发布版本

确认一切就绪后，可以准备该应用程序的发布版本。

1. 在主菜单中，依次选择“生成”>“清理解决方案”，删除上一生成过程中创建的中间文件和输出文件。 此步骤不是必需的，但它可以清理调试生成输出。

2. 使用工具栏（当前显示“调试”）上的下拉列表控件把 HelloWPFApp 的生成配置从“调试”更改为“发布” 。

3. 选择“生成”>“生成解决方案”来生成解决方案 。

恭喜你完成本教程！ 可在解决方案和项目目录 (...\HelloWPFApp\HelloWPFApp\bin\Release) 下找到生成的 .exe 文件。

后续步骤

恭喜你完成本教程！ 若要了解详情，请继续学习后续教程。

继续学习更多 WPF 教程

请参阅

- 工作效率提示

反馈

此页面是否有帮助? 是 否

在 Visual Studio 中使用 C# 创建 Windows 窗体应用

项目 • 2023/05/08

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

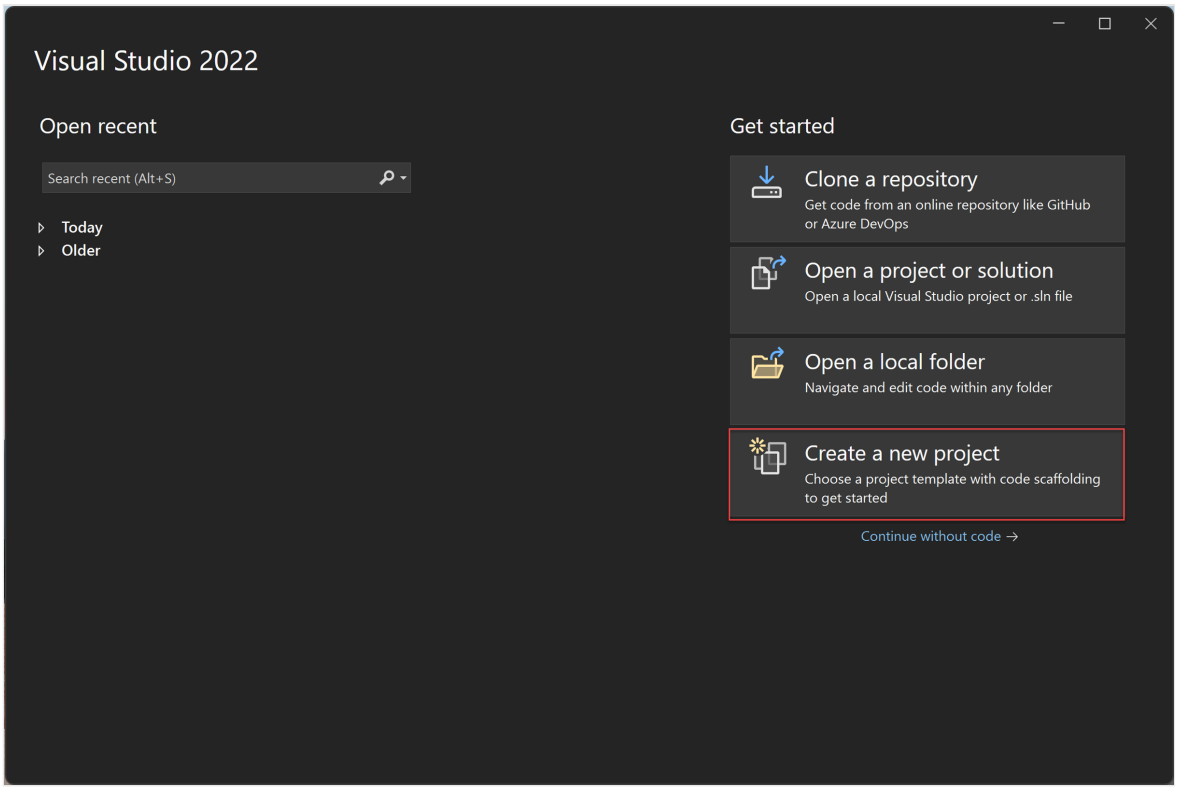
在本教程中，你将创建一个具有基于 Windows 的用户界面 (UI) 的简单 C# 应用程序。

如果尚未安装 Visual Studio，请转到 Visual Studio 2022 下载 页免费安装。

创建项目

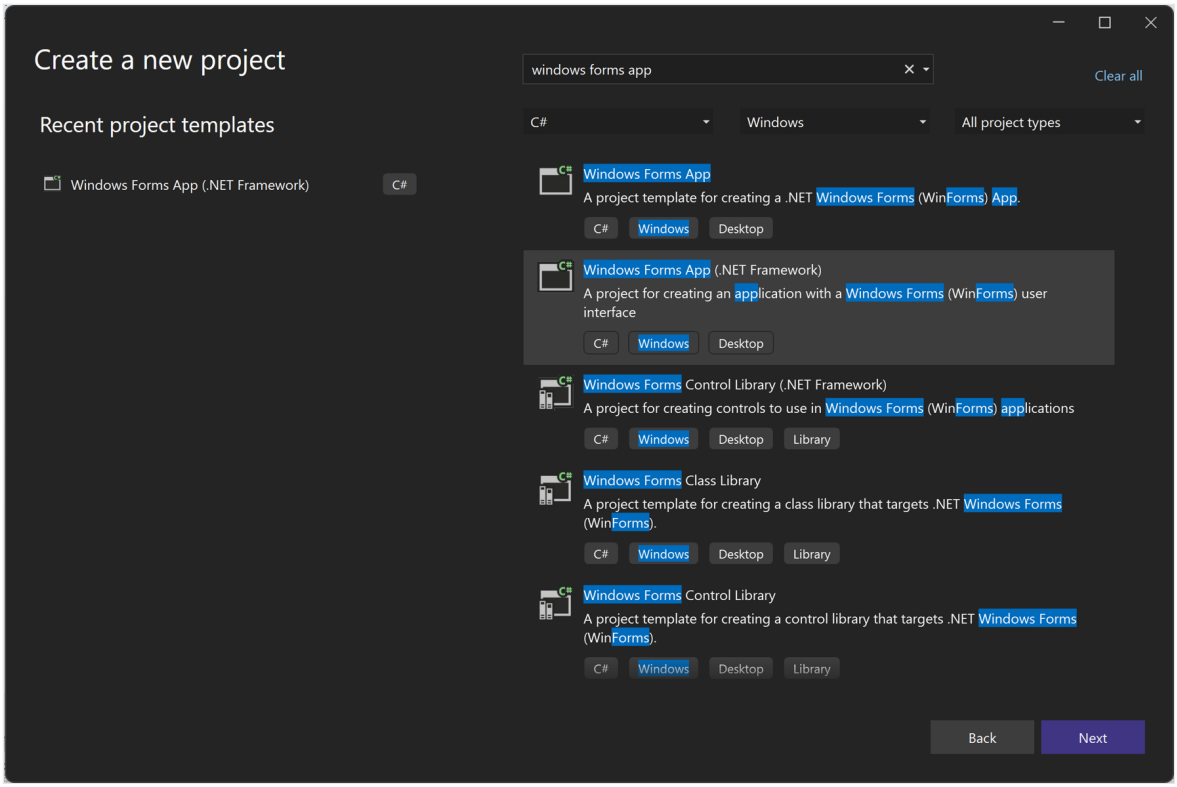
首先，创建 C# 应用程序项目。 项目类型随附了所需的全部模板文件，无需添加任何内容。

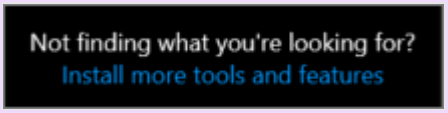
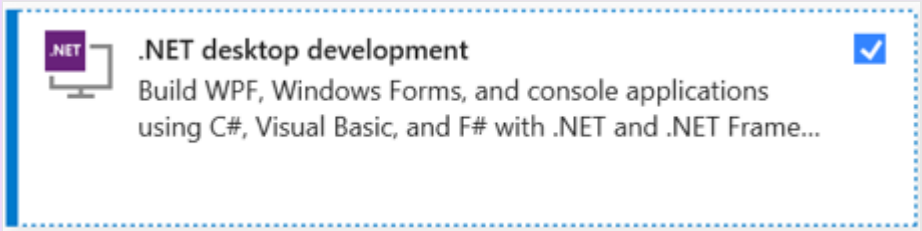
1. 打开 Visual Studio。
2. 在“开始”窗口中，选择“创建新项目”。



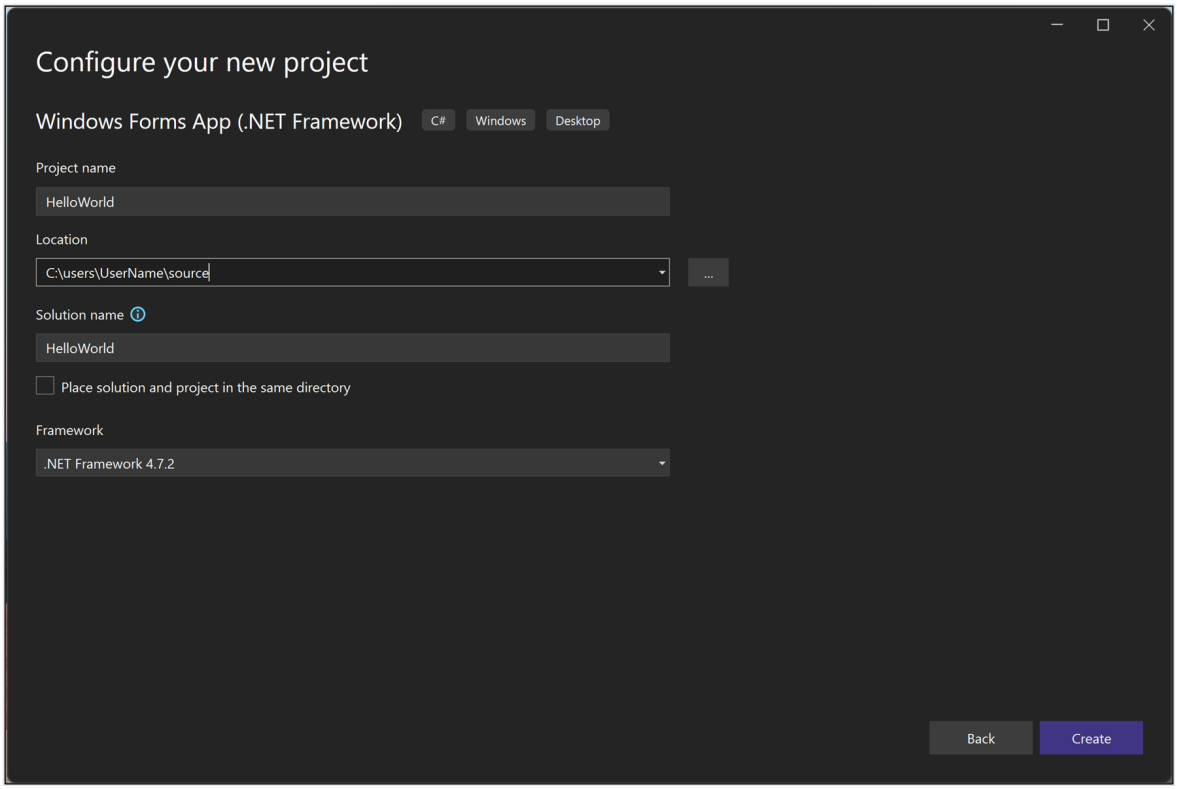
3. 在“创建新项目”窗口中，为 C# 选择“Windows 窗体应用(.NET Framework)”模板 。

   （如果愿意，可以优化搜索以快速访问所需的模板。例如，在搜索框中输入或键入“Windows 窗体应用”。接下来，从“语言”列表中选择“C#”，然后从“平台”列表中选择“Windows”。）



> **① 备注**
>
> 如果未看到“Windows 窗体应用(.NET Framework)” 模板，则可以通过“创建新项目” 窗口安装该模板。 在“找不到所需内容?”消息中，选择“安装更多工具和功能”链接 。
>
> 
>
>
> 接下来，在 Visual Studio 安装程序中，选择“.NET 桌面开发”工作负载。
>
> 
>
>
> 之后，在 Visual Studio 安装程序中选择“修改”按钮。 系统可能会提示你保存所有内容；如果出现提示，请按照指示进行操作。 接下来，选择“继续”以安装工作负载。 然后，返回到“**创建项目**”过程中的步骤 2。

4. 在“配置新项目”窗口中，在“项目名称”框中键入或输入“HelloWorld” 。 然后选择“创建”。



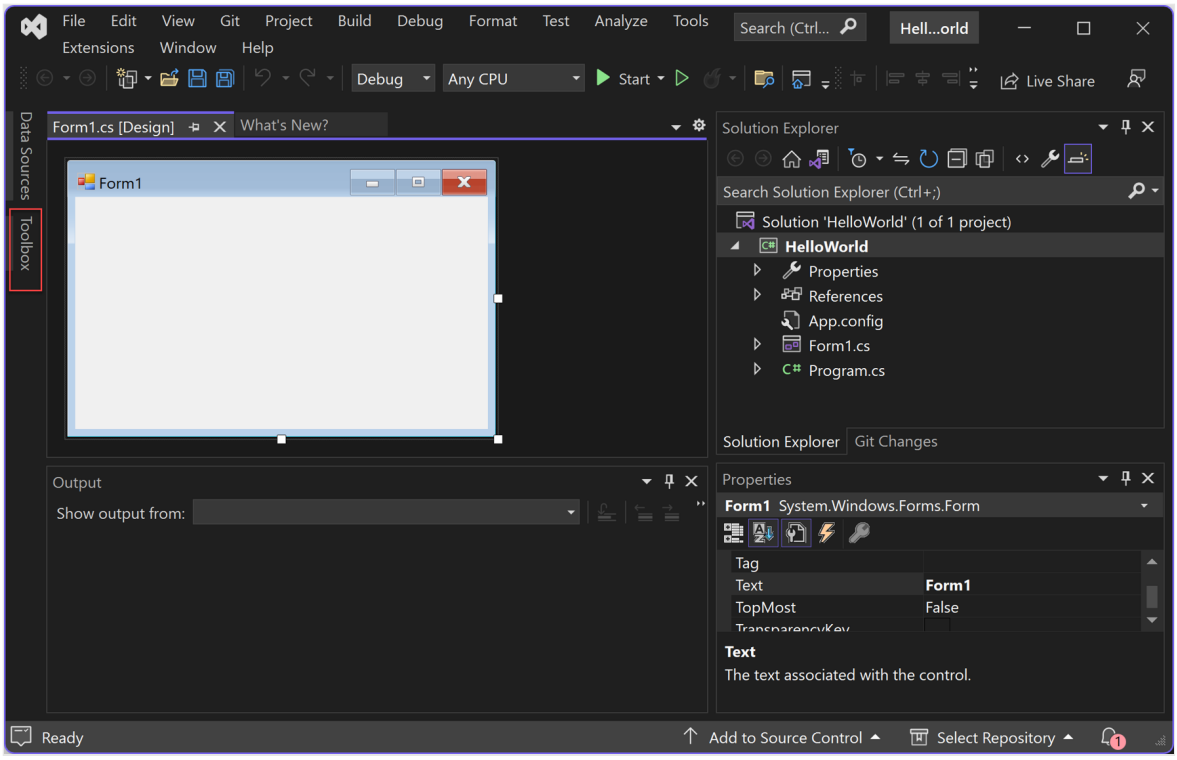
此时，Visual Studio 将打开新项目。

创建应用程序

选择 C# 项目模板并为文件命名后，Visual Studio 会打开一个窗体。 窗体就是 Windows 用户界面。 通过向窗体添加控件创建“Hello World”应用程序，然后运行该应用程序。

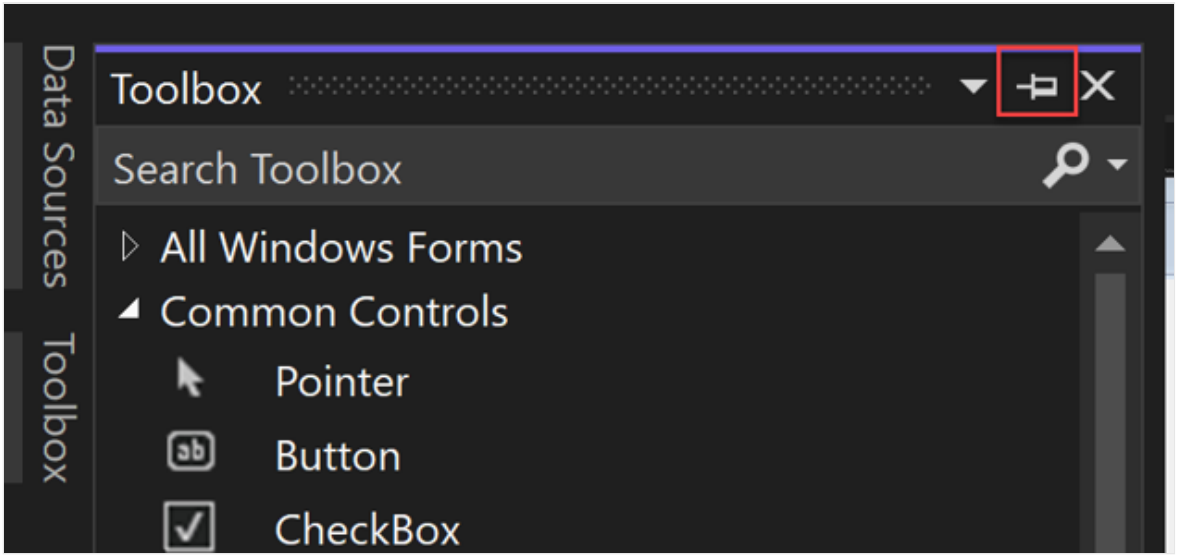
向窗体添加按钮

1. 选择“工具箱”，打开“工具箱”弹出窗口。

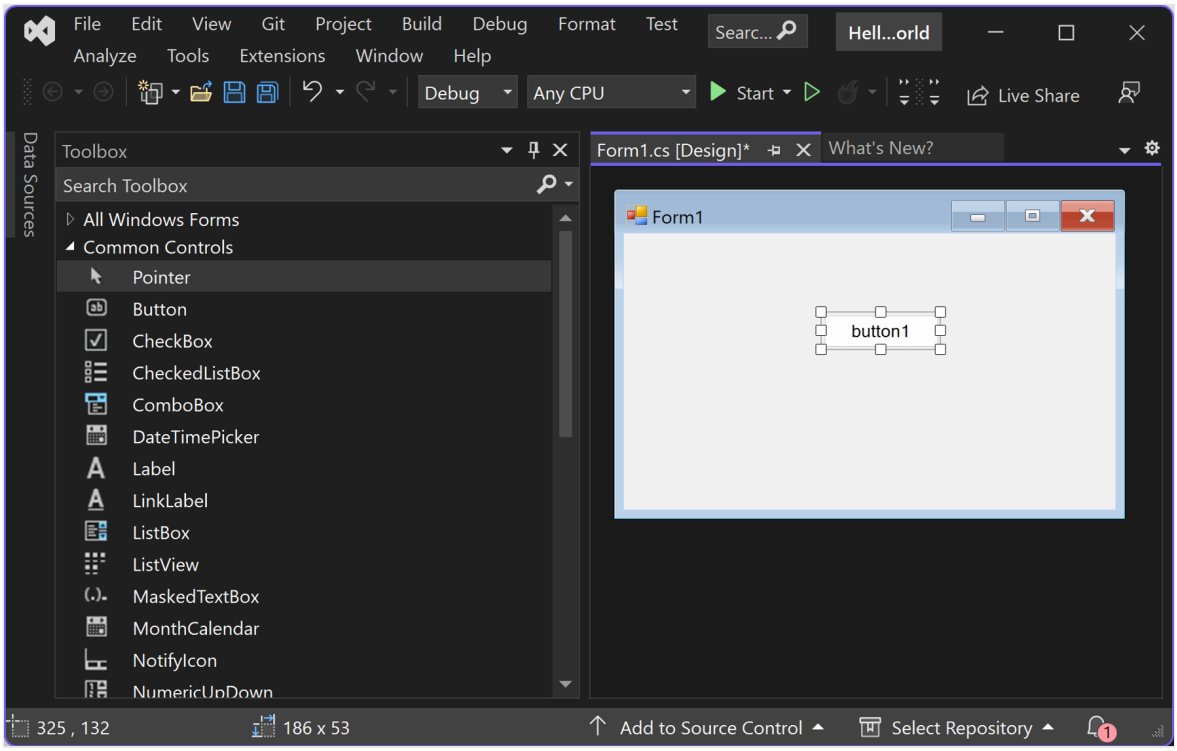


（如果没有看到“工具箱”浮出控件选项，可以从菜单栏中打开它。为此，请选择“查看”>“工具箱”。或者，按 Ctrl+Alt+X。）

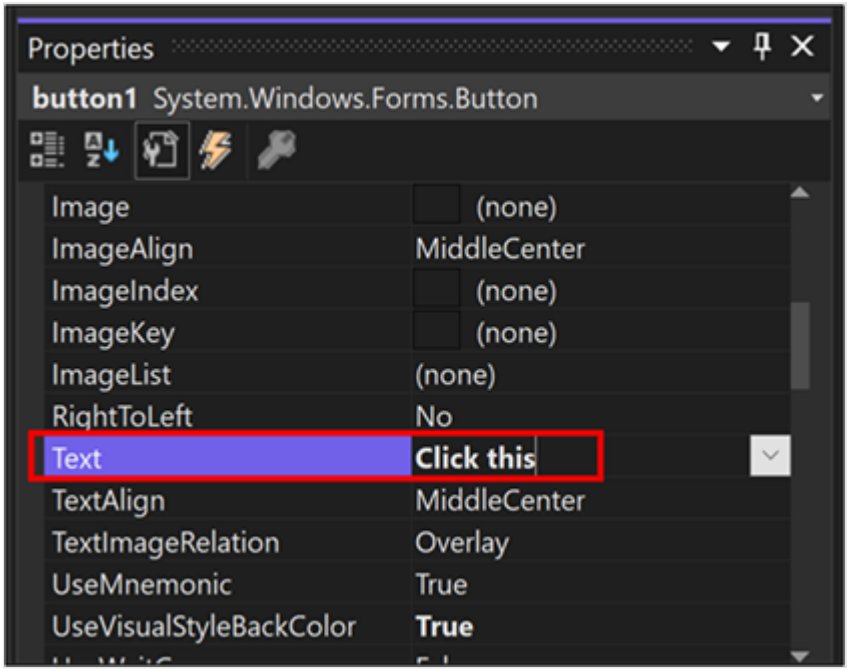
2. 展开“常见控件”，然后选择“固定”图标以停靠“工具箱”窗口。



3. 选择“按钮”控件，然后将其拖到窗体上。

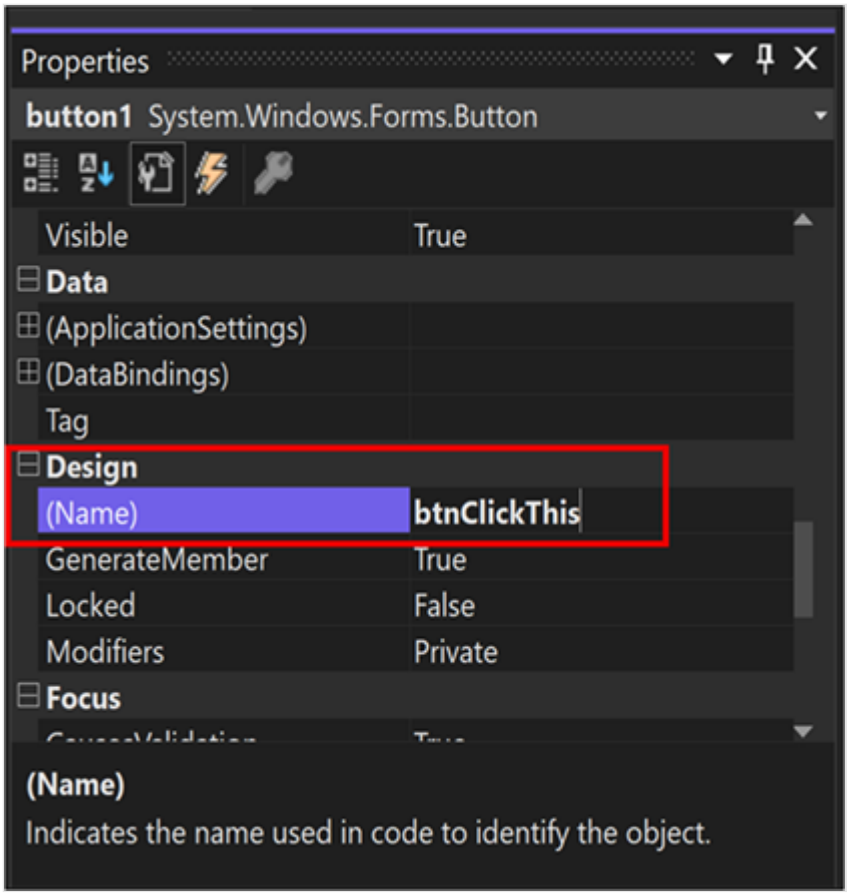


4. 在属性窗口中，找到“文本”，将名称从“Button1”更改为 `Click this` 然后按 Enter 。



（如果没有看到“属性”窗口，可以从菜单栏打开它。为此，请选择“查看”>“属性窗口”。或者，按 F4。）

5. 在属性窗口的“设计”部分，将名称从“Button1”更改为 `btnClickThis`，然后按 Enter 。



> ⓘ **备注**
>
> 如果按字母顺序排列了属性窗口列表，则 button1 会显示在 (DataBindings) 部分中 。

向窗体添加标签

添加创建操作的按钮控件后，我们来添加发送文本的标签控件。

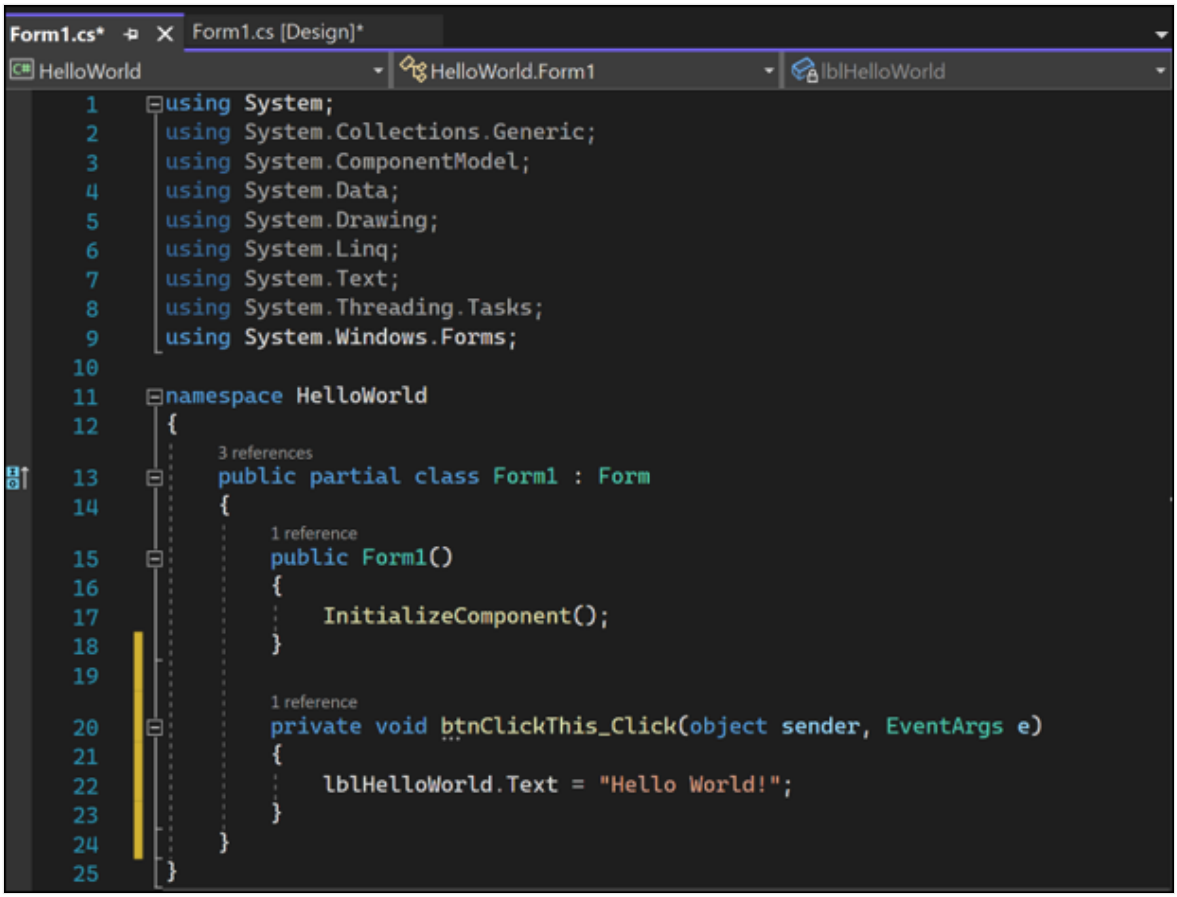
1. 从“工具箱”窗口选择“标签”控件，然后将其拖到窗体上，并放到“单击此处”按钮下方 。
2. 在属性窗口的“设计”部分或“(DataBindings)”部分，将 label1 的名称更改为 `lblHelloWorld`，然后按 Enter 。

向窗体添加代码

1. 在“Form1.cs [设计]”窗口中，双击“单击此处”按钮，打开“Form1.cs”窗口。

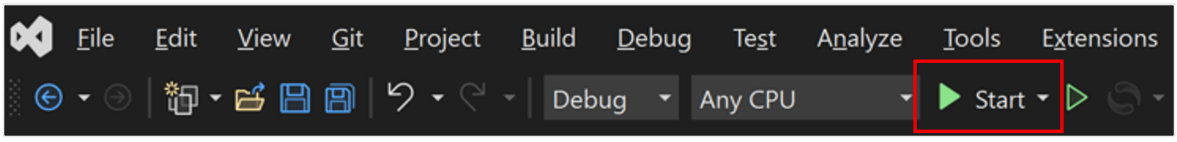
   （或者，可在“解决方案资源管理器” 中展开“Form1.cs” ，然后选择“Form1” 。）

2. 在“Form1.cs” 窗口中，在“private void” 行后，键入或输入 `lblHelloWorld.Text = "Hello World!";`，如以下屏幕截图所示：



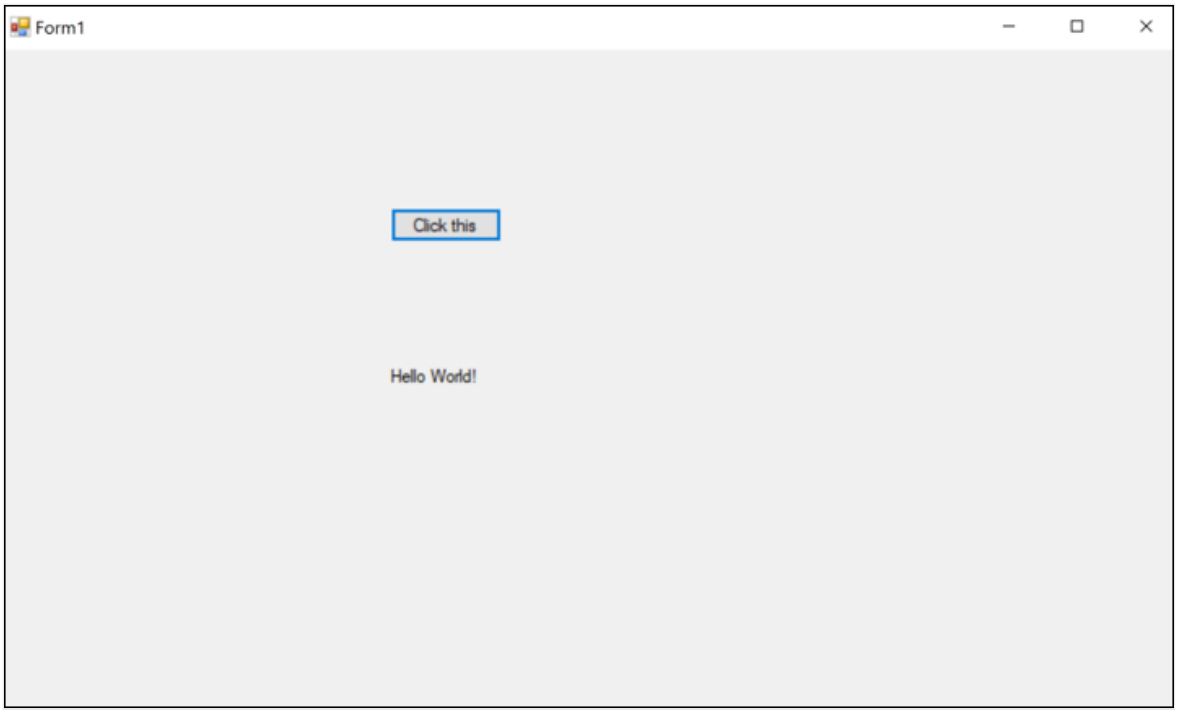
运行应用程序

1. 选择“启动”按钮运行应用程序。



将出现以下几种情况。 在 Visual Studio IDE 中，“诊断工具”窗口打开，同时还会打开一个“输出”窗口 。 在 IDE 外部，会出现一个“Form1”对话框 。 其中包含“单击此处”按钮和显示“label1”的文本 。

2. 选择“Form1”对话框中的“单击此处”按钮 。 请注意，“label1”文本会更改为“Hello World!” 。



3. 关闭“Form1” 对话框以停止运行应用。

后续步骤

恭喜你完成本教程。 要更加深入地了解，请继续学习下面的教程：

教程：创建图片查看器

请参阅

- 更多 C# 教程
- Visual Basic 教程
- C++ 教程

教程：在 Visual Studio 中创建一个图片查看器 Windows 窗体应用

项目 • 2023/06/22

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在此系列的三个教程中，你将创建一个 Windows 窗体应用程序，用于加载和显示图片。 Visual Studio 集成设计环境 (IDE) 提供了创建应用所需的工具。 若要了解详细信息，请参阅欢迎使用 Visual Studio IDE。

在第一个教程中，你将了解如何：

- ✓ 创建一个使用 Windows 窗体的 Visual Studio 项目
- ✓ 添加布局元素
- ✓ 运行应用程序

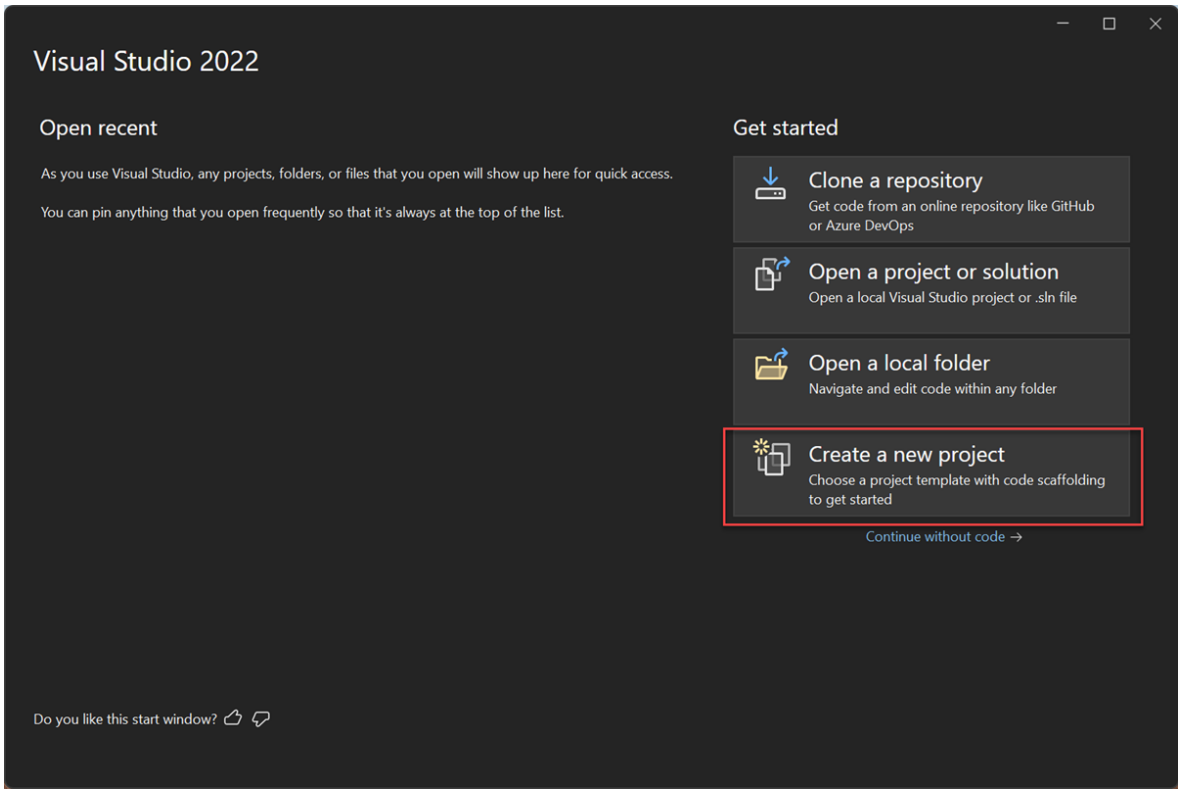
先决条件

若要完成本教程，必须具有 Visual Studio。 请访问 Visual Studio 下载页 获取免费版本。

创建 Windows 窗体项目

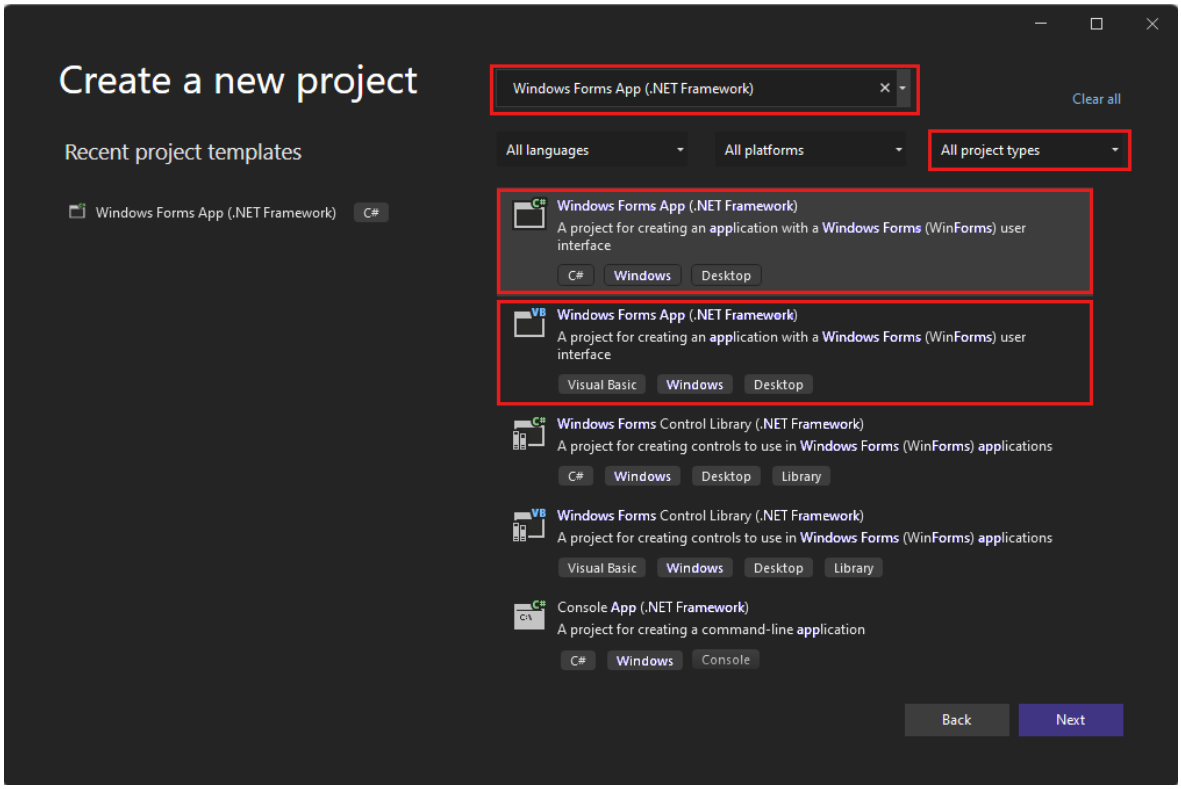
创建图片查看器时，第一步是创建 Windows 窗体应用项目。

1. 打开 Visual Studio。
2. 在“开始”窗口中，选择“创建新项目”。



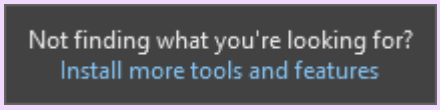
3. 在“创建新项目”窗口中，搜索“Windows 窗体”。 然后，从“项目类型”列表中选择“桌面”。

4. 针对 C# 或 Visual Basic，选择“Windows 窗体应用(.NET Framework)”模板，然后选择“下一步”。

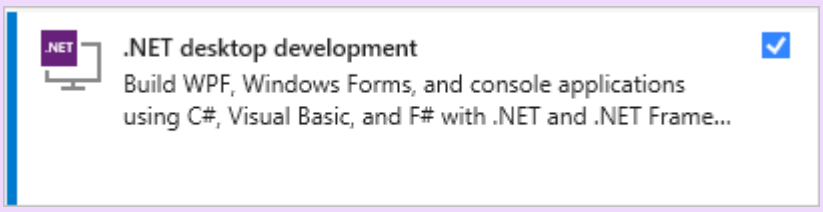


备注

如果未看到"Windows 窗体应用(.NET Framework)"模板，则可以通过"创建新项目"窗口安装该模板。 在"找不到所需内容?"消息中，选择"安装更多工具和功能"链接 。

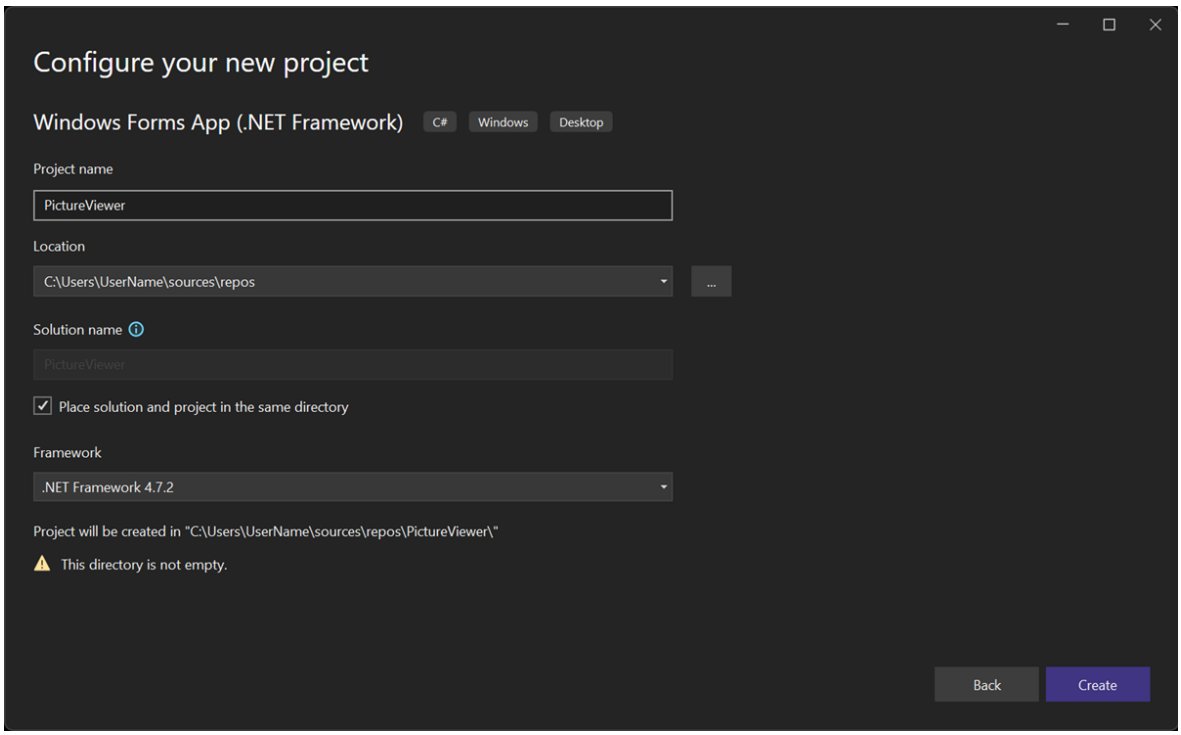


接下来，在 Visual Studio 安装程序中，选择".NET 桌面开发"。



在 Visual Studio 安装程序中，选择"修改"。 系统可能会提示你保存工作内容。 接下来，选择"继续"以安装工作负载。

5. 在"配置新项目"窗口中，将项目命名为"PictureViewer"，然后选择"创建"。



Visual Studio 将为你的应用创建解决方案。 解决方案是应用所需全部项目和文件的容器。

此时，Visual Studio 在 Windows 窗体设计器中显示一个空窗体。

添加布局元素

图片查看应用包含一个图片框、一个复选框和四个按钮，你将在下一个教程中添加它们。 布局元素控制其在窗体上的位置。 此部分演示如何更改窗体的标题、调整窗体大小以及添加布局元素。

1. 在项目中，选择“Windows 窗体设计器”。 对于 C#，该选项卡显示 Form1.cs [Design]，对于 Visual Basic 则显示 Form1.vb [Design] 。

2. 选择 Form1 中的任意位置。

3. 属性窗口现在显示窗体的各个属性。 属性窗口通常位于 Visual Studio 的右下角。 此部分控制各种属性，例如前景色和背景色、显示在窗体顶部的标题文本以及窗体的大小。

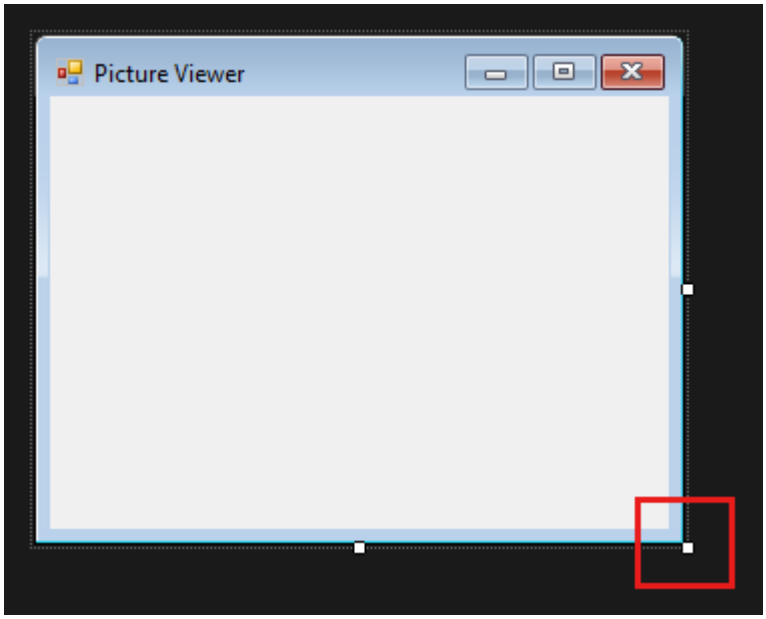
   如果看不到“属性”，请选择“查看”>“属性窗口” 。

4. 在属性窗口中找到“Text”属性 。 根据列表排序的方式，您可能需要向下滚动。 输入值“图片查看器”，然后按 Enter 键。

   窗体的标题栏中现在出现文本“图片查看器”。

   > **备注**
   >
   > 可以按类别或字母顺序显示属性。 使用属性窗口中的按钮来回切换。

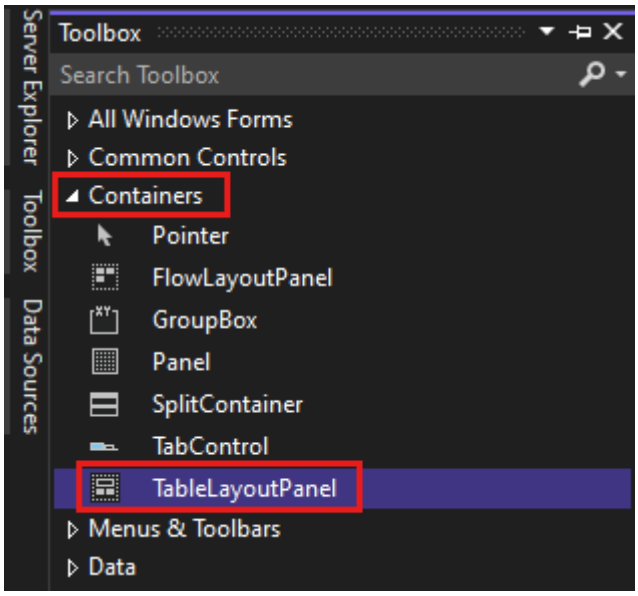
5. 再次选择窗体。 选择窗体右下角的拖动图柄。 该图柄是窗体右下角的一个白色小正方形。

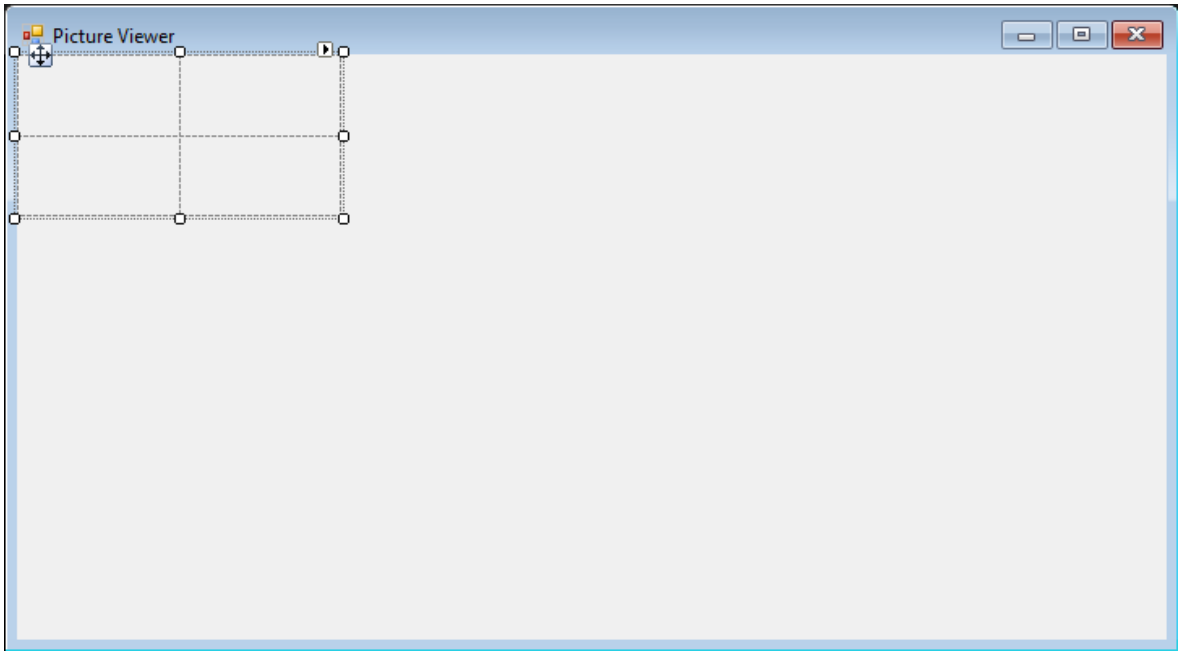
   拖动手柄以调整窗体的大小，使其更宽且更高一些。 如果查看属性窗口，你会发现“Size”属性已更改 。 还可通过更改“Size”属性来更改窗体的大小。

6. 在 Visual Studio IDE 的左侧，选择“工具箱”选项卡。如果没有看到，则从菜单栏中选择“查看”>“工具箱”或者按 Ctrl+Alt+X 键 。

7. 选择容器旁边的小三角形符号以打开该组。

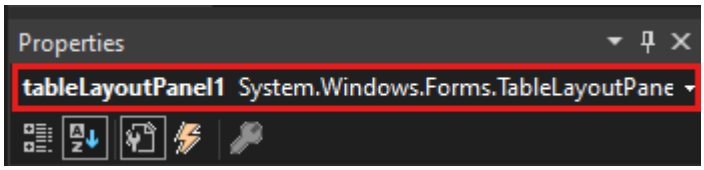


8. 双击“工具箱”中的“TableLayoutPanel” 。 你也可以将控件从工具箱拖动到窗体上。TableLayoutPanel 控件将显示在窗体中。

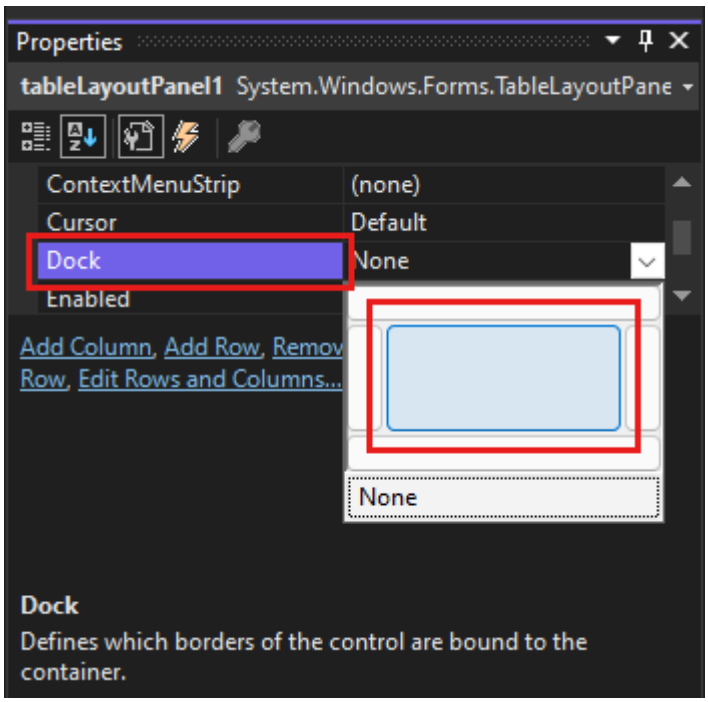


> **备注**
>
> 添加 TableLayoutPanel 后，如果窗体中出现标题为“TableLayoutPanel 任务”的窗口，请点击窗体内的任何位置来关闭此窗口。

9. 选择“TableLayoutPanel”。 可以通过查看属性窗口来验证选择了什么控件。



10. 选中“TableLayoutPanel”后，找到“Dock”属性，其值为“无” 。 选择下拉箭头，然后选择“填充”，即下拉菜单中间的大按钮。

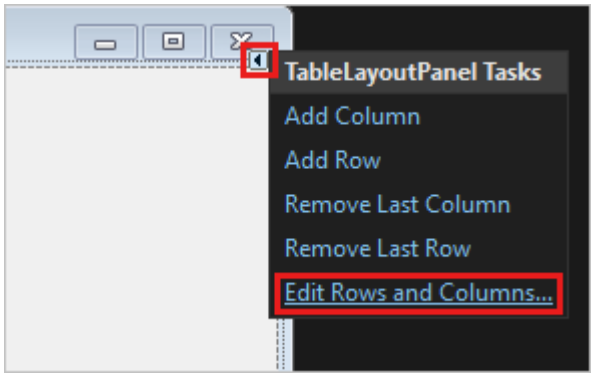


“停靠”是指窗口连接另一个窗口或区域所用的一种方式。

TableLayoutPanel 现在填充整个窗体。 如果再次调整窗体的大小，则 TableLayoutPanel 将保持停靠状态，并自行调整大小以适合窗体。

11. 在窗体中，选择“TableLayoutPanel”。 右上角有一个黑色的小三角形按钮。

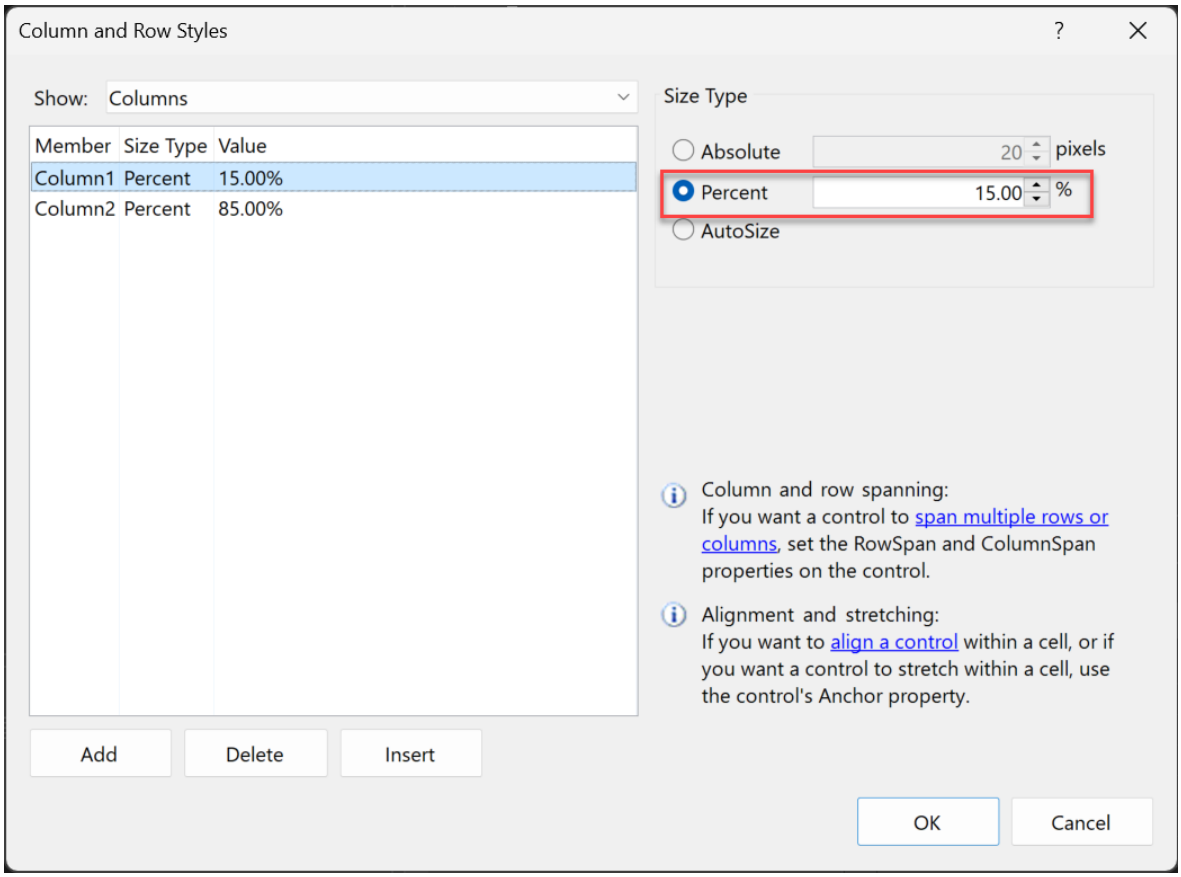
选择该三角形以显示控件的任务列表。



12. 选择“编辑行和列”，以显示“列和行样式”对话框 。

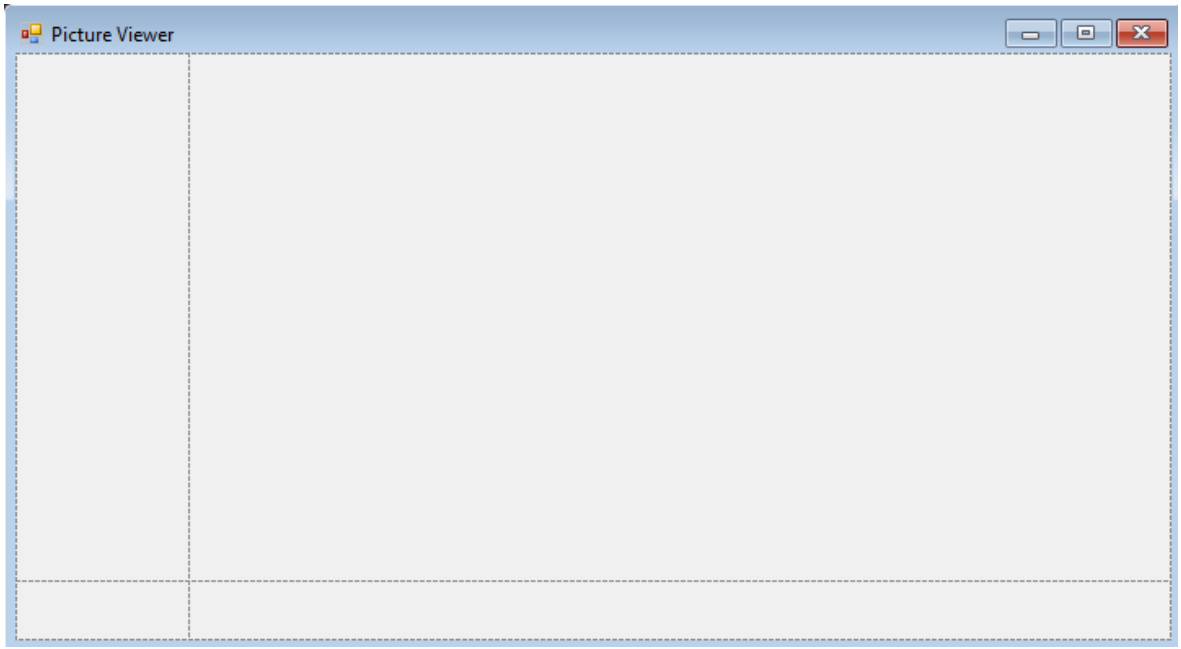
13. 选择“Column1”，将其大小设置为 15%。 确保已选中“百分比”按钮。

14. 选择“Column2”并将其设置为 85%。



15. 在“列和行样式”对话框顶部的“显示”中，选择“行” 。 将“Row1”设置为 90% 并将“Row2”设置为 10%。 选择“确定”，保存所做更改。

    TableLayoutPanel 现在具有一个大的顶部行、一个小的底部行、一个小的左侧列和一个大的右侧列。



布局已完成。

> **备注**

在运行应用程序之前，请通过选择“全部保存”工具栏按钮来保存应用。 或者，要保存应用，请从菜单栏中选择“文件”>“全部保存”（或按 Ctrl+Shift+S 键）。 最佳做法是尽早且经常保存。

运行应用

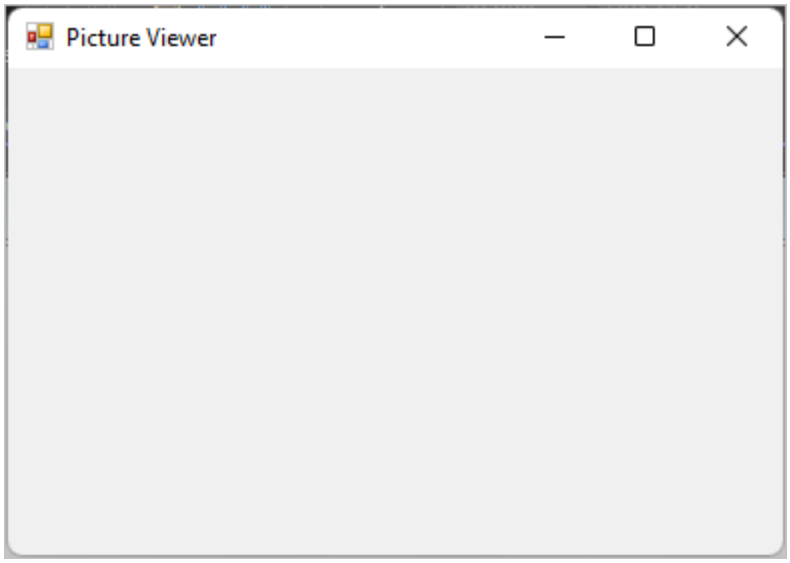
创建 Windows 窗体应用项目时，会生成一个运行的程序。 在此阶段，图片查看器应用不会执行太多操作。 它目前只是显示了一个在标题栏中显示“图片查看器”的空窗口。

若要运行应用，请执行以下步骤。

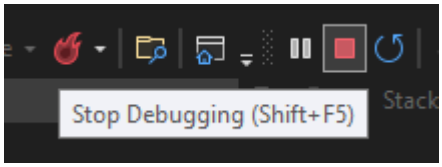
1. 使用下列方法之一：

   - 选择 F5 键。
   - 在菜单栏上，依次选择“调试”>“开始调试”。
   - 在工具栏上，选择“开始”按钮。

   Visual Studio 运行应用。 此时将显示标题为“图片查看器”的窗口。

   查看 Visual Studio IDE 工具栏。 运行应用程序时，工具栏上将显示更多按钮。 利用这些按钮，你可执行停止和启动应用之类的操作，并帮助你跟踪到任何错误。

2. 使用以下其中一种方法停止应用：

   - 在工具栏上，选择“停止调试”按钮。
   - 在菜单栏上，选择“调试”>“停止调试”。

- 在键盘上，按 Shift+F5 键 。
- 选择“图片查看器”窗口上角的“X”按钮 。

当你从 Visual Studio IDE 内部运行应用时，该过程称为调试。 运行应用程序以查找和修复 bug。 您可以执行相同的过程来运行和调试其他程序。 要了解有关调试的详细信息，请参阅初探调试器。

后续步骤

请继续阅读下一篇教程，了解如何向图片查看器程序添加控件。

教程第 2 部分：向图片查看器添加控件

教程：向 Visual Studio 中的图片查看器 Windows 窗体应用添加 UI 控件

项目 • 2023/06/19

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在此系列的三个教程中，你将创建一个 Windows 窗体应用程序，用于加载和显示图片。 Visual Studio 集成设计环境 (IDE) 提供了创建应用所需的工具。 若要了解详细信息，请参阅欢迎使用 Visual Studio IDE。

此程序有一个图片框、一个复选框和几个按钮，你可以用来控制应用程序。 此教程介绍如何添加这些控件。

在第二个教程中，你将了解如何：

- ✓ 向应用程序添加控件
- ✓ 在布局面板中添加按钮
- ✓ 更改控件名称和位置
- ✓ 添加对话框组件

先决条件

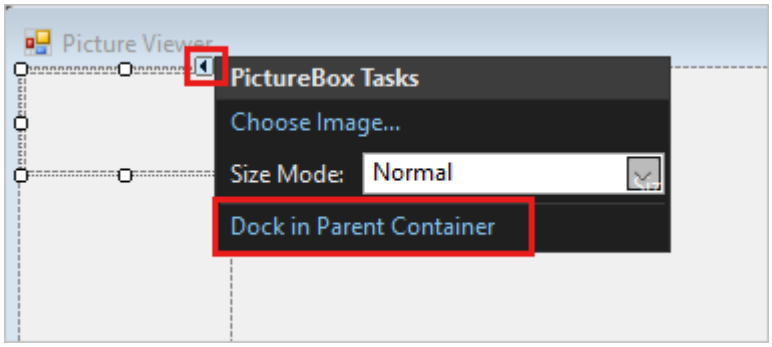
此教程基于上一教程创建图片查看器应用程序。 如果尚未学习此教程，请先完成此教程。

向应用程序添加控件

图片查看器应用程序使用 PictureBox 控件来显示图片。 它使用一个复选框和多个按钮来管理图片和背景，以及关闭应用。 你将在 Visual Studio IDE 中从“工具箱”中添加 PictureBox 和复选框。

1. 打开 Visual Studio。 图片查看器项目显示在“打开最近使用的文件”下。
2. 在 Windows 窗体设计器中，选择在前面的教程中添加的 TableLayoutPanel。 检查 tableLayoutPanel1 是否显示在属性窗口中 。
3. 在 Visual Studio IDE 的左侧，选择“工具箱”选项卡。如果没有看到，则从菜单栏中选择“查看”>“工具箱”或者按 Ctrl+Alt+X 键 。 在“工具箱”中，展开“公共控件”。
4. 双击“PictureBox”，将 PictureBox 控件添加到窗体。 Visual Studio IDE 会将 PictureBox 控件添加到 TableLayoutPanel 的第一个空单元格中。

5. 选择新的 PictureBox 控件以将其选中，然后选择新 PictureBox 控件上的黑色三角形以显示其任务列表。



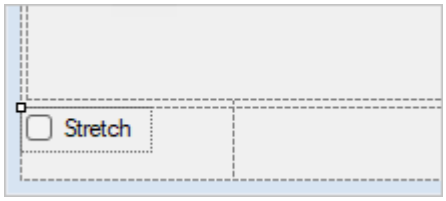
6. 选择“在父容器中停靠”，这会将 PictureBox 的“Dock”属性设置为“填充” 。 可以在属性窗口中查看该值。

7. 在 PictureBox 的属性窗口中，将“ColumnSpan”属性设置为“2” 。 PictureBox 现在填充这两列。

8. 将“BorderStyle”属性设置为“Fixed3D” 。

9. 在 Windows 窗体设计器中选择“TableLayoutPanel” 。 然后，在“工具箱”中双击“CheckBox”项，以添加新的 CheckBox 控件 。 由于 PictureBox 占据了 TableLayoutPanel 中的前两个单元格，因此 CheckBox 控件将添加到左下方的单元格。

10. 选择“Text”属性，然后输入“拉伸” 。



在布局面板中添加按钮

到目前为止，控件已添加到 TableLayoutPanel 中。 以下步骤演示了如何在 TableLayoutPanel 中向新的布局面板添加四个按钮。

1. 选择窗体上的“TableLayoutPanel”。 打开“工具箱”，选择“容器” 。 双击“FlowLayoutPanel”，将新控件添加到 TableLayoutPanel 的最后一个单元格。

2. 将 FlowLayoutPanel 的“Dock”属性设置为“填充” 。 可以通过选择黑色三角形，然后选择“在父容器中停靠”来设置此属性。

   FlowLayoutPanel 是一个容器，它将其他控件按顺序排列在一行中。

3. 选择新的“FlowLayoutPanel”，然后打开“工具箱”并选择“公共控件” 。 双击“按钮”项以添加一个名为“button1”的按钮控件 。

4. 再次双击“按钮”以添加其他按钮。 IDE 将调用下一个“button2”。

5. 以这种方式再添加两个按钮。 另一种方法是选择“button2”，然后选择“编辑”>“复制”或按 Ctrl+C 键 。 接下来，在菜单栏上，选择“编辑”>“粘贴”或按 Ctrl+V 键 。 粘贴按钮的副本。 此时再次粘贴该副本。 请注意，IDE 会将“button3”和“button4”添加到 FlowLayoutPanel。

6. 选择第一个按钮，并将其“Text”属性设置为“显示图片” 。

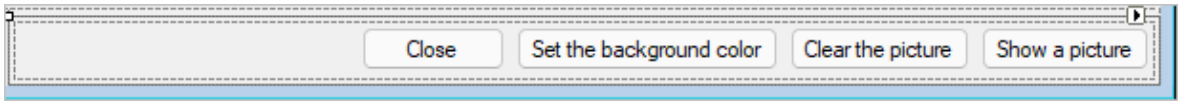
7. 分别将后面三个按钮的“Text”属性设置为“清除图片”、“设置背景色”和“关闭” 。

8. 若要调整按钮的大小并进行排列，请选择“FlowLayoutPanel”。 将“FlowDirection”属性设置为“RightToLeft” 。

   这些按钮会自行与单元格的右侧对齐，并颠倒其顺序，以使“显示图片”按钮位于右侧。 可以在 FlowLayoutPanel 周围拖动按钮，按任意顺序排列它们。

9. 选择“关闭”按钮以将其选中。 然后，要同时选择其余按钮，请在按住 Ctrl 键的同时选择它们。

10. 在属性窗口中，将“AutoSize”属性设置为“True” 。 调整按钮大小以适应其文本。



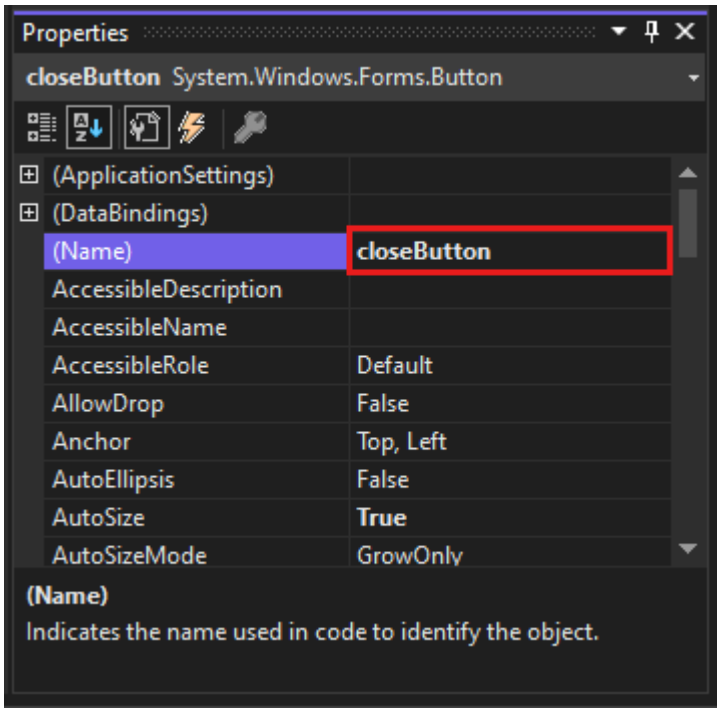
你可以运行程序以了解控件的外观。 选择 F5 键，选择“调试”>“开始调试”，或选择“开始”按钮 。 你添加的按钮还未执行任何操作。

更改控件名称

窗体上有四个按钮，在 C# 中的名称分别为“button1”、“button2”、“button3”和“button4” 。 在 Visual Basic 中，任何控件名称的第一个字母都默认大写，所以这些按钮的名称为“Button1”、“Button2”、“Button3”和“Button4” 。 使用以下步骤为它们指定包含更多信息的名称。

1. 在窗体上，选择“关闭” 按钮。 如果你仍选择了所有按钮，请按 Esc 键取消选择。

2. 在属性窗口中查找“(名称)” 。 将名称更改为“closeButton”。



IDE 不接受包含空格的名称。

3. 将其他三个按钮重命名为“backgroundButton” 、 “clearButton”和“showButton” 。 你可通过选择“属性” 窗口中的控件选择器下拉列表来验证这些名称。 新的按钮名称将出现。

可以更改任何控件的名称，如 TableLayoutPanel 或复选框。

添加对话框组件

你的应用可以通过组件打开图片文件并选择背景色。 组件类似于控件。 使用“工具箱”向窗体添加组件。 使用属性窗口设置其属性。

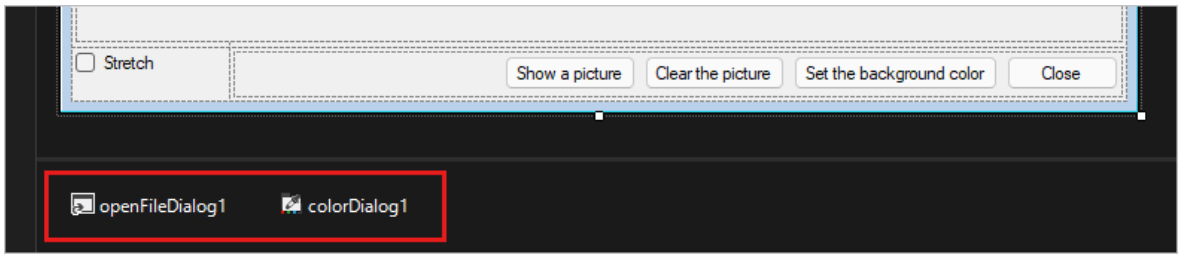
与控件不同，向窗体添加组件不会添加可见项。 相反，这将提供可使用代码触发的某些行为。 例如，组件可打开“打开文件”对话框。

在此部分，你将向窗体添加 OpenFileDialog 组件和 ColorDialog 组件。

1. 选择 Windows 窗体设计器（“Form1.cs [Design]”） 。 然后打开“工具箱”并选择“对话框”组 。

2. 双击“OpenFileDialog”向窗体添加一个名为“openFileDialog1”的组件 。

3. 双击“ColorDialog”，添加一个名为 colorDialog1 的组件 。 该组件在 Windows 窗体设计器的底部显示为图标。



4. 选择“openFileDialog1”图标并设置以下两个属性：

   - 将“Filter”属性设置为以下值：

     控制台

     ```
     JPEG Files (*.jpg)|*.jpg|PNG Files (*.png)|*.png|BMP Files
     (*.bmp)|*.bmp|All files (*.*)|*.*
     ```

   - 将 Title 属性设置为以下内容：“选择一个图片文件”

   “Filter”属性设置指定“选择图片”对话框显示的类型 。

后续步骤

请继续阅读下一篇教程，了解如何向应用程序添加代码。

教程第 3 部分：向图片查看器添加代码

教程：向 Visual Studio 中的图片查看器 Windows 窗体应用添加代码

项目 • 2023/03/21

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在此系列的三个教程中，你将创建一个 Windows 窗体应用程序，用于加载和显示图片。 Visual Studio 集成设计环境 (IDE) 提供了创建应用所需的工具。 若要了解详细信息，请参阅欢迎使用 Visual Studio IDE。

控件使用 C# 或 Visual Basic 代码执行与其相关的操作。

在第三个教程中，你将了解如何：

- ✓ 为控件添加事件处理程序
- ✓ 编写代码以打开对话框
- ✓ 为其他控件编写代码
- ✓ 运行应用程序

先决条件

此教程基于前面的教程创建图片查看器应用程序和向图片查看器添加 UI 控件。 如果尚未学习这些教程，请先完成这些教程。

为控件添加事件处理程序

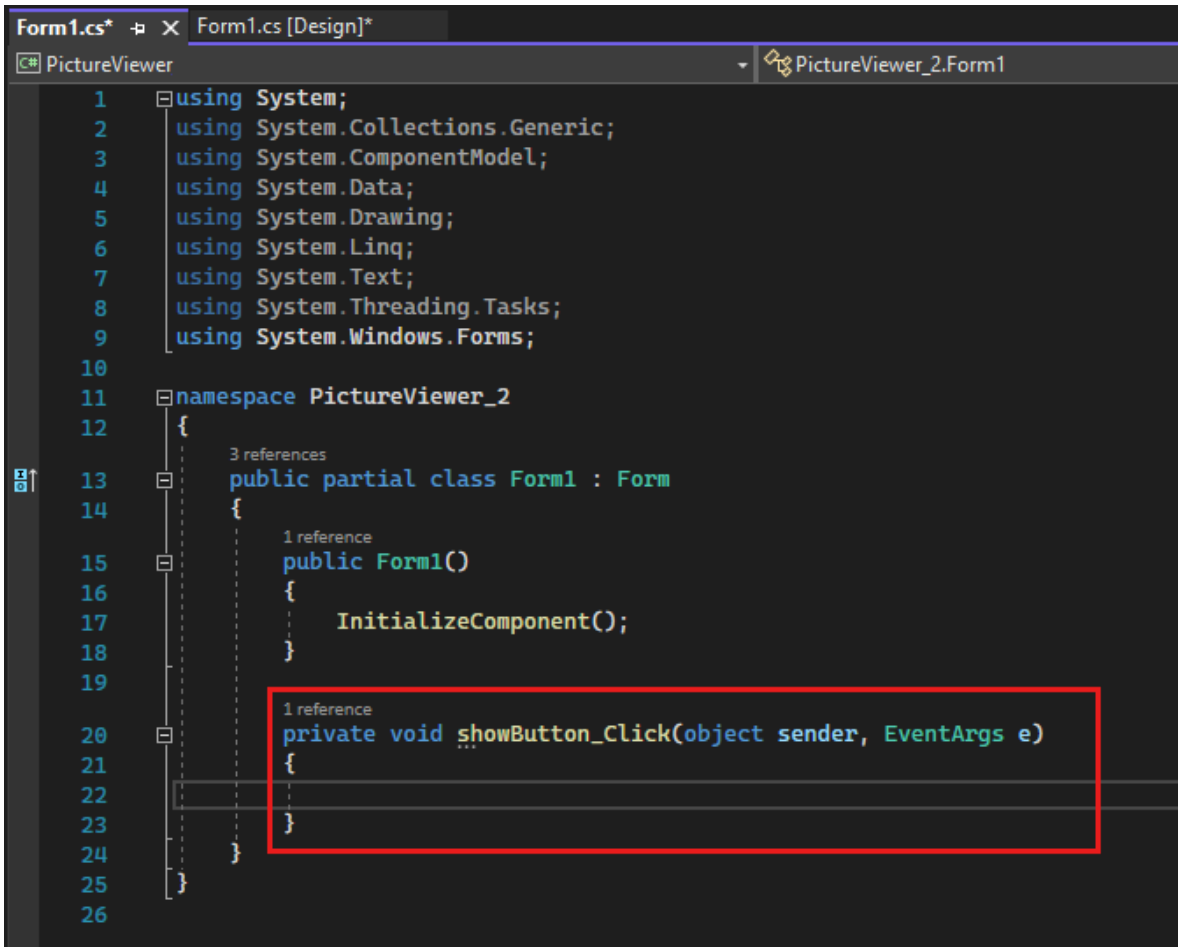
在此部分，为第二个教程向图片查看器应用程序添加控件中添加的控件添加事件处理程序。 发生操作（如选择按钮）时，应用程序会调用事件处理程序。

1. 打开 Visual Studio。 图片查看器项目显示在“打开最近使用的文件”下。

2. 在 Windows 窗体设计器中，并双击“显示图片”按钮 。 或者，可以选择窗体上的“显示图片”按钮，然后按 Enter 键 。

   Visual Studio IDE 会在主窗口中打开一个选项卡。 在 C# 中，选项卡名为 Form1.cs。 如果使用的是 Visual Basic，则该选项卡名为 Form1.vb。

   此选项卡显示窗体后面的代码文件。



> **① 备注**
>
> Form1.vb 选项卡可能会将“showButton”显示为“ShowButton” 。

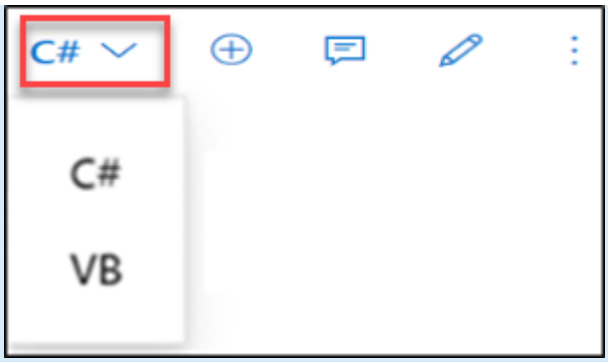
3. 重点考虑这一部分的代码。

C#

C#

```
private void showButton_Click(object sender, EventArgs e)
{
}
```

> **ⓘ 重要**
>
> 使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



4. 再次选择“Windows 窗体设计器”选项卡，然后双击“清除图片”按钮以打开其代码 。 对于剩余两个按钮，重复此操作。 Visual Studio IDE 每次都会向窗体的代码文件添加一个新方法。

5. 双击 Windows 窗体设计器中的 CheckBox 控件以添加 `checkBox1_CheckedChanged()` 方法 。 选中或清除复选框时，它将调用此方法。

   下面代码片段显示了你在代码编辑器中看到的新代码。

C#

C#

```
private void clearButton_Click(object sender, EventArgs e)
{
}

private void backgroundButton_Click(object sender, EventArgs e)
{
}

private void closeButton_Click(object sender, EventArgs e)
{
}

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
}
```

方法（包括事件处理程序）可以根据你的需要命名。 使用 IDE 添加事件处理程序时，IDE 将基于控件的名称和正在处理的事件创建一个名称。

例如，名为 showButton 的按钮的 Click 事件称为 `showButton_Click()` 或 `ShowButton_Click()`。 如果要更改代码的变量名，请右键单击代码中的变量，然后选择

“重构”>“重命名” 。 将重命名代码中变量的所有实例。 有关更多信息，请参阅重命名重构。

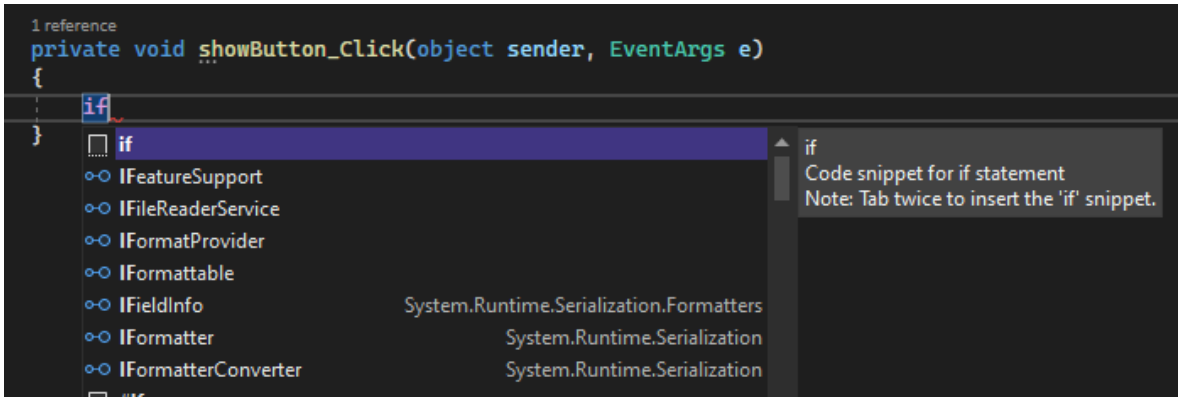
编写代码以打开对话框

“显示图片”按钮使用 OpenFileDialog 组件显示图片文件。 此过程添加用于调用该组件的代码。

Visual Studio IDE 提供了一个名为 IntelliSense 的强大工具。 当你在键入时，IntelliSense 会建议可能的代码。

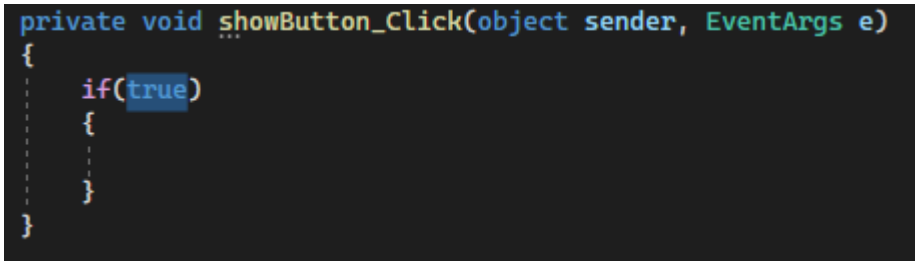
1. 在 Windows 窗体设计器中，并双击“显示图片”按钮 。 IDE 将光标移到 `showButton_Click()` 或 `ShowButton_Click()` 方法内。

2. 在两个大括号 `{ }` 之间或 `Private Sub...` 和 `End Sub` 之间的空行上键入一个 i。 这时会打开一个 IntelliSense 窗口。



3. “IntelliSense”窗口应该会突出显示 `if` 一词。 选择 Tab 键以插入 `if`。

4. 选择“true”，然后针对 C# 键入 `op` 以进行覆盖，或者针对 Visual Basic 键入 `Op` 以进行覆盖。



IntelliSense 显示 openFileDialog1。

5. 选择 Tab 键以添加 openFileDialog1。

6. 在 openFileDialog1 后直接键入句点 (`.`) 或点号。 IntelliSense 提供 OpenFileDialog 组件的所有属性和方法。 开始键入 `ShowDialog` 并选择 Tab 键。 `ShowDialog()` 方法

将显示“打开文件”对话框。

7. 在 `ShowDialog` 中的“g”后立即添加括号 `()`。 代码应该为 `openFileDialog1.ShowDialog()`。

8. 对于 C#，添加一个空格，然后添加两个等于号 (`==`)。 对于 Visual Basic，添加一个空格，然后使用单个等号 (`=`)。

9. 添加另一个空格。 使用 IntelliSense 输入 DialogResult。

10. 键入一个点，在 IntelliSense 窗口中打开 DialogResult 值。 输入字母 `O`，然后选择 Tab 键插入“OK”。

> **备注**
>
> 第一行代码应会完成。 对于 C#，输出应如下所示。
>
> ```
> if (openFileDialog1.ShowDialog() == DialogResult.OK)
> ```
>
> 对于 Visual Basic，此代码行应如下所示。
>
> ```
> If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
> ```

11. 添加以下代码行。

C#

```
C#

pictureBox1.Load(openFileDialog1.FileName);
```

可以复制并粘贴或使用 IntelliSense 添加这些代码行。 最终的 `showButton_Click()` 方法应与以下代码类似。

C#

```
C#

private void showButton_Click(object sender, EventArgs e)
{
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
```

```
    }
}
```

1. 将下面的注释添加到代码中。

C#

```
C#

private void showButton_Click(object sender, EventArgs e)
{
    // Show the Open File dialog. If the user clicks OK, load the
    // picture that the user chose.
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        pictureBox1.Load(openFileDialog1.FileName);
    }
}
```

最佳做法是始终注释代码。 通过代码注释，可以在将来更轻松地理解和维护代码。

为其他控件编写代码

如果现在运行应用程序，可以选择“显示图片”。 图片查看器将打开“打开文件”对话框，你可在其中选择要显示的图片。

在此部分，为其他事件处理程序添加代码。

1. 在 Windows 窗体设计器中，并双击“清除图片”按钮 。 在大括号中添加代码。

C#

```
C#

private void clearButton_Click(object sender, EventArgs e)
{
    // Clear the picture.
    pictureBox1.Image = null;
}
```

2. 双击“设置背景色”按钮，并在大括号中添加代码。

C#

```
C#

private void backgroundButton_Click(object sender, EventArgs e)
{
    // Show the color dialog box. If the user clicks OK, change the
    // PictureBox control's background to the color the user chose.
    if (colorDialog1.ShowDialog() == DialogResult.OK)
        pictureBox1.BackColor = colorDialog1.Color;
}
```

3. 双击“关闭”按钮，并在大括号中添加代码。

C#

```
C#

private void closeButton_Click(object sender, EventArgs e)
{
    // Close the form.
    this.Close();
}
```

4. 双击“拉伸”复选框，并在大括号中添加代码。

C#

```
C#

private void checkBox1_CheckedChanged(object sender, EventArgs e)
{
    // If the user selects the Stretch check box,
    // change the PictureBox's
    // SizeMode property to "Stretch". If the user clears
    // the check box, change it to "Normal".
    if (checkBox1.Checked)
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
    else
        pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
}
```

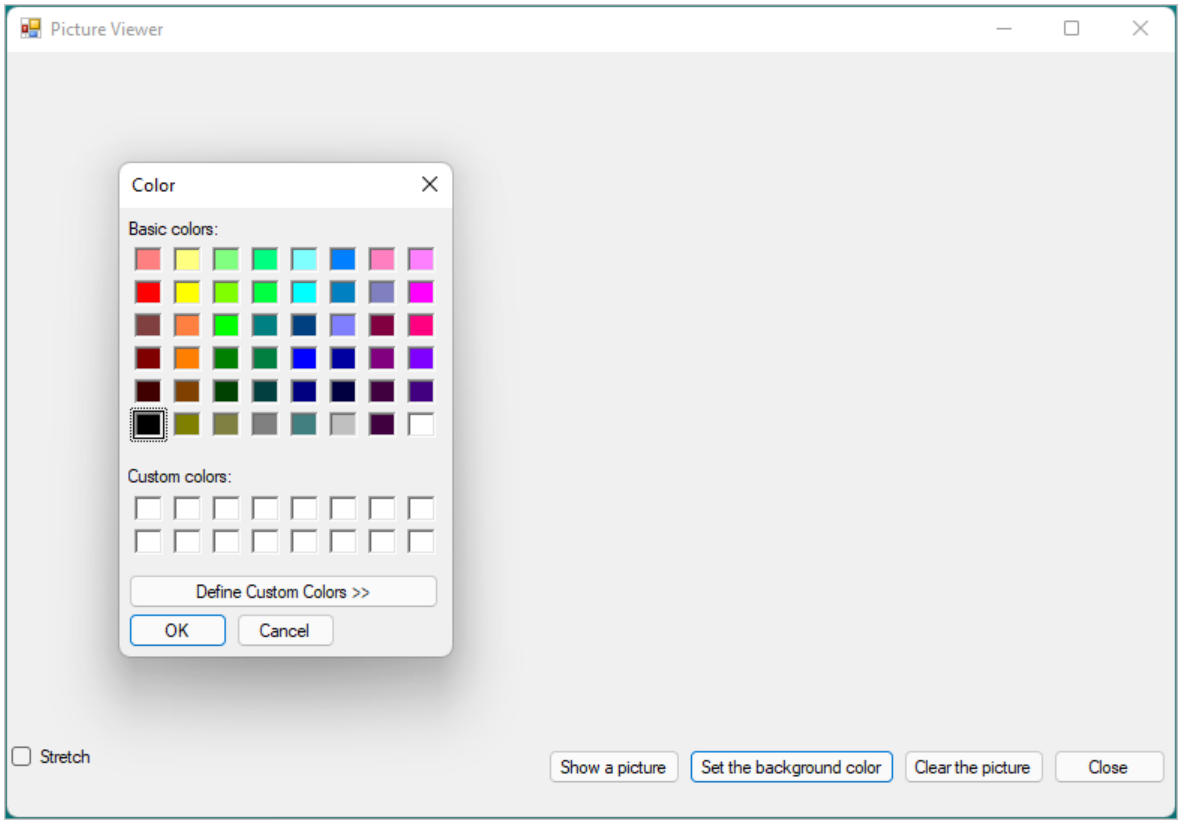
运行应用程序

编写应用程序时，你随时都可以运行该应用程序。 在前面部分添加代码后，图片查看器将完成。 与前面的教程一样，使用以下方法之一运行应用程序：

- 选择 F5 键。
- 在菜单栏上，依次选择“调试”>“开始调试” 。
- 在工具栏上，选择“开始”按钮。

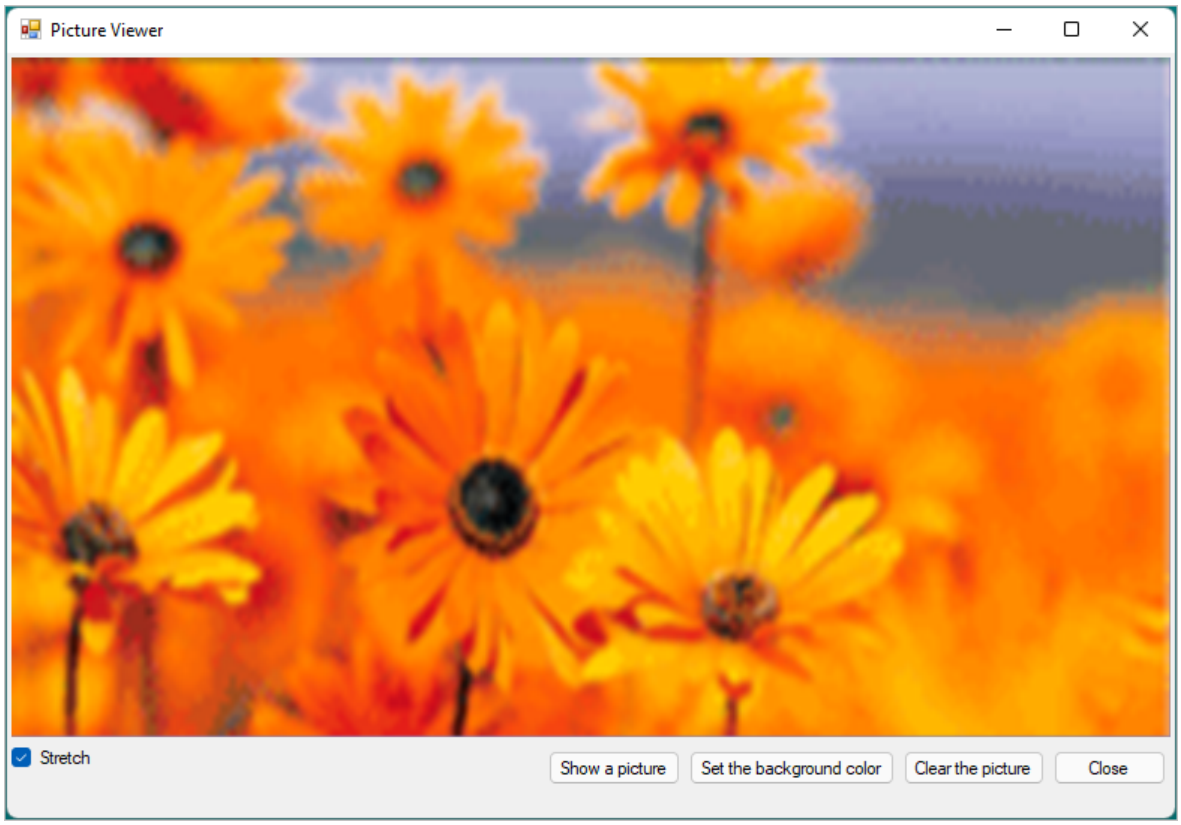
此时将显示标题为“图片查看器”的窗口。 测试所有控件。

1. 选择“设置背景色”按钮。 随即打开“颜色”对话框。



2. 选择一种颜色来设置背景色。

3. 选择“显示图片”以显示图片。



4. 选择“拉伸”，然后取消选择。

5. 选择“清除图片”按钮以确保显示内容消失。

6. 选择“关闭”以退出应用。

后续步骤

祝贺！ 你已完成该系列教程。 你已在 Visual Studio IDE 中完成以下编程和设计任务：

- 创建了一个使用 Windows 窗体的 Visual Studio 项目
- 为图片查看应用程序添加了布局
- 添加了按钮和复选框
- 添加了对话框
- 为控件添加了事件处理程序
- 编写了 C# 或 Visual Basic 代码来处理事件

继续学习另一个教程系列，了解如何创建计时数学测验。

教程 2：创建计时数学测验

教程：创建数学测验 WinForms 应用

项目 • 2023/06/19

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本系列的四个教程中，你将创建一个数学测验。 测验包含四个随机数学问题，测验者需要尝试在指定的时间内回答这些问题。

Visual Studio 集成开发环境 (IDE) 提供了创建应用所需的工具。 若要详细了解此 IDE，请参阅欢迎使用 Visual Studio IDE。

在第一个教程中，你将了解如何：

- ✓ 创建一个使用 Windows 窗体的 Visual Studio 项目。
- ✓ 向窗体添加标签、按钮和其他控件。
- ✓ 设置控件的属性。
- ✓ 保存并运行项目。

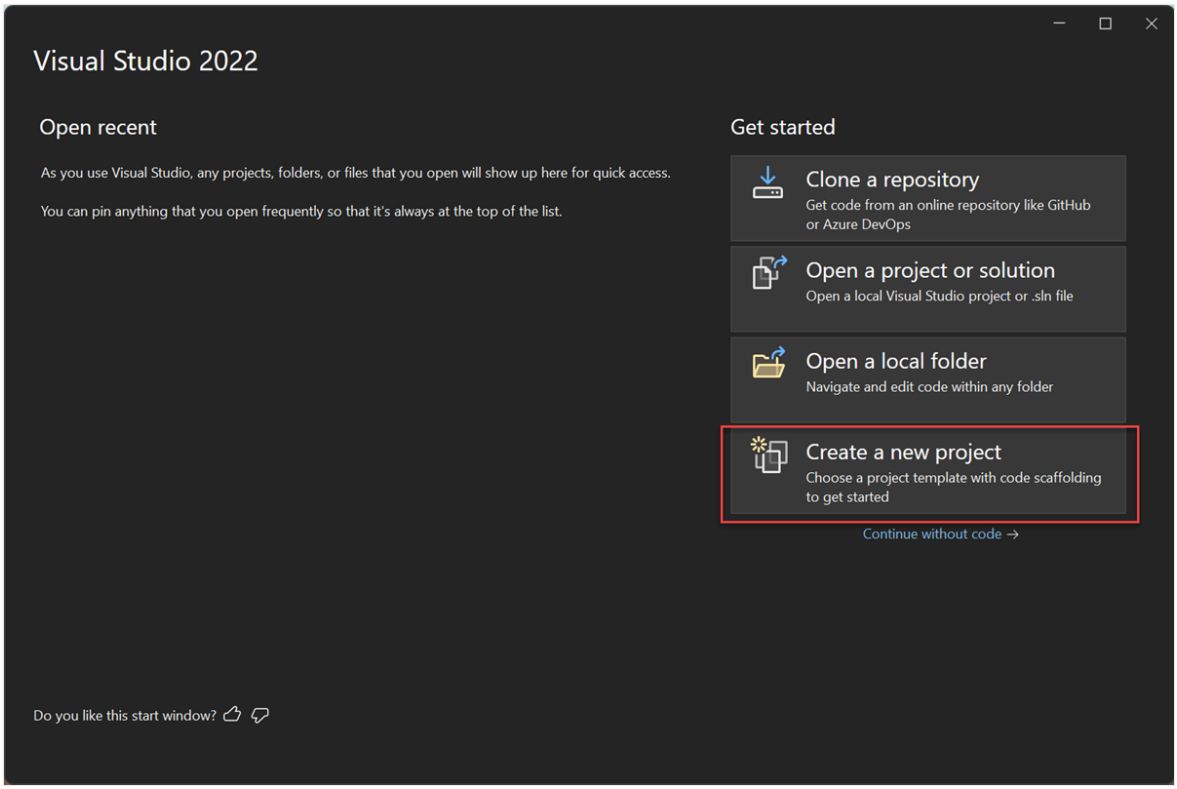
先决条件

若要完成本教程，必须具有 Visual Studio。 请访问 Visual Studio 下载页 获取免费版本。

创建 Windows 窗体项目

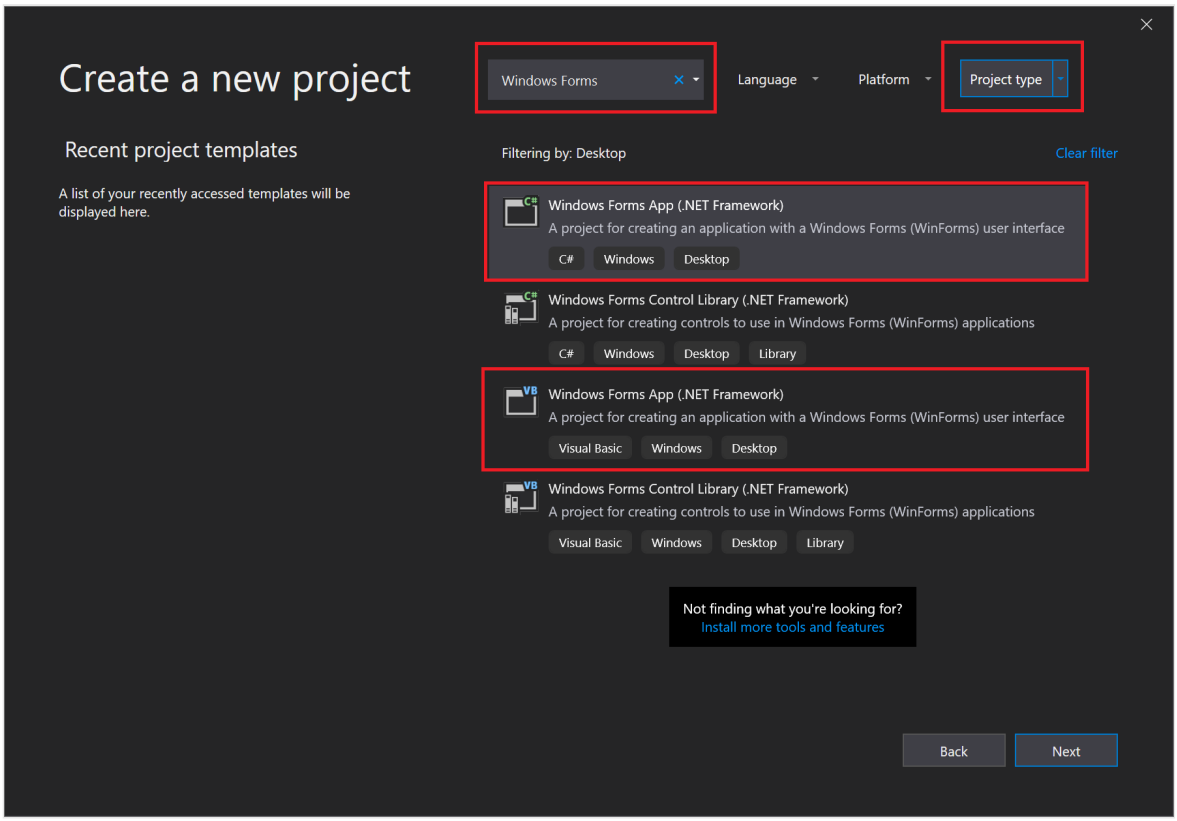
创建数学测验时，第一步是创建 Windows 窗体应用项目。

1. 打开 Visual Studio。
2. 在“开始”窗口中，选择“创建新项目”。



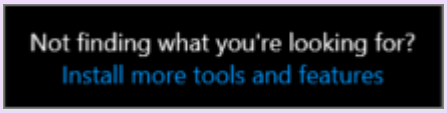
3. 在“创建新项目”窗口中，搜索“Windows 窗体” 。 然后，从“项目类型”列表中选择“桌面” 。

4. 针对 C# 或 Visual Basic，选择“Windows 窗体应用(.NET Framework)”模板，然后选择“下一步” 。

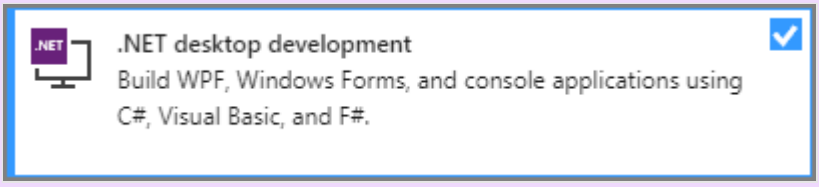


① 备注

如果未看到“Windows 窗体应用(.NET Framework)”模板，则可以通过“创建新项目”窗口安装该模板。 在“找不到所需内容?”消息中，选择“安装更多工具和功能” 。



接下来，在 Visual Studio 安装程序中，选择“.NET 桌面开发”。



在 Visual Studio 安装程序中，选择“修改”。 系统可能会提示你保存工作内容。 接下来，选择“继续”以安装工作负载。

5. 在“配置新项目”窗口中，将项目命名为“MathQuiz”，然后选择“创建” 。

Visual Studio 将为你的应用创建解决方案。 解决方案是你的应用所需的全部项目和文件的容器。

设置窗体属性

选择模板并为文件命名后，Visual Studio 会打开一个窗体。 本部分演示如何更改某些窗体属性。

1. 在项目中，选择“Windows 窗体设计器”。 对于 C#，设计器选项卡显示 Form1.cs [Design]，对于 Visual Basic 则显示 Form1.vb [Design] 。

2. 选择窗体“Form1”。

3. 属性窗口现在显示窗体的各个属性。 此窗体通常位于 Visual Studio 的右下角。 如果看不到“属性”，请选择“查看”>“属性窗口” 。

4. 在属性窗口中找到“Text”属性 。 根据列表排序的方式，您可能需要向下滚动。 对于“文本”值输入“数学测验”，然后按 Enter 。

   窗体的标题栏中现在包含文本“数学测验”。

> **① 备注**
>
> 可以按类别或字母顺序显示属性。 使用属性窗口中的按钮来回切换。

5. 将窗体大小更改为 500 像素（宽）× 400 像素（高）。

   可以通过拖动窗体的边缘或拖动图柄来调整窗体的大小，直到“属性”窗口中的“Size”值显示为适当的大小 。 拖动图柄是窗体右下角的一个白色小正方形。 还可通过更改“Size”属性的值来更改窗体的大小。

6. 将“FormBorderStyle”属性的值更改为“Fixed3D”，并将“MaximizeBox”属性设置为“False”。

   这些值用于防止测验对象改变窗体大小。

创建“剩余时间”框

数学测验的右上角包含一个框。 该框显示测验剩余的秒数。 本部分演示如何对该框使用标签。 你将在本系列后面的教程中添加倒计时计时器的代码。

1. 在 Visual Studio IDE 的左侧，选择“工具箱”选项卡。如果没有看到工具箱，则从菜单栏中选择“查看”>“工具箱”或者按 Ctrl+Alt+X 键 。

2. 选择“工具箱”中的 Label 控件，然后将其拖到窗体上。

3. 在“属性”框中，为标签设置以下属性：

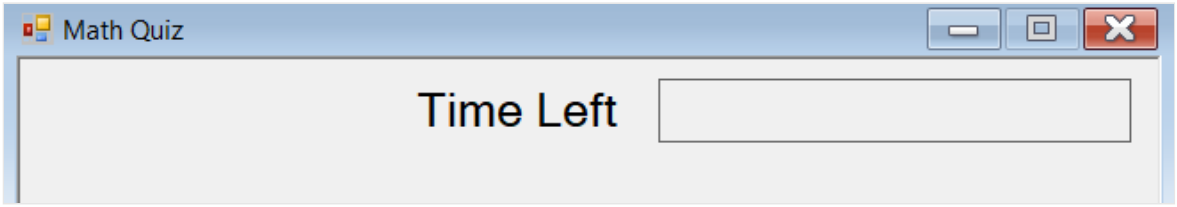
   - 将“(Name)”设置为 timeLabel 。
   - 将“AutoSize”更改为“False”，这样你便可以调整框的大小 。
   - 将“BorderStyle”更改为“FixedSingle”以在框的周围绘制线条 。
   - 将“Size”设置为“200, 30” 。
   - 选择“Text”属性，然后选择 Backspace 键以清除“Text”值 。
   - 选择“字体”属性旁边的加号 (+)，然后将“大小”设置为“15.75”。

4. 将标签移动到窗体的右上角。 当出现蓝色空格线时，使用它们将控件放置在窗体上。

5. 从“工具箱”中再添加一个“标签”控件，然后将其字号设置为“15.75”。

6. 将此标签的“Text”属性设置为“剩余时间” 。

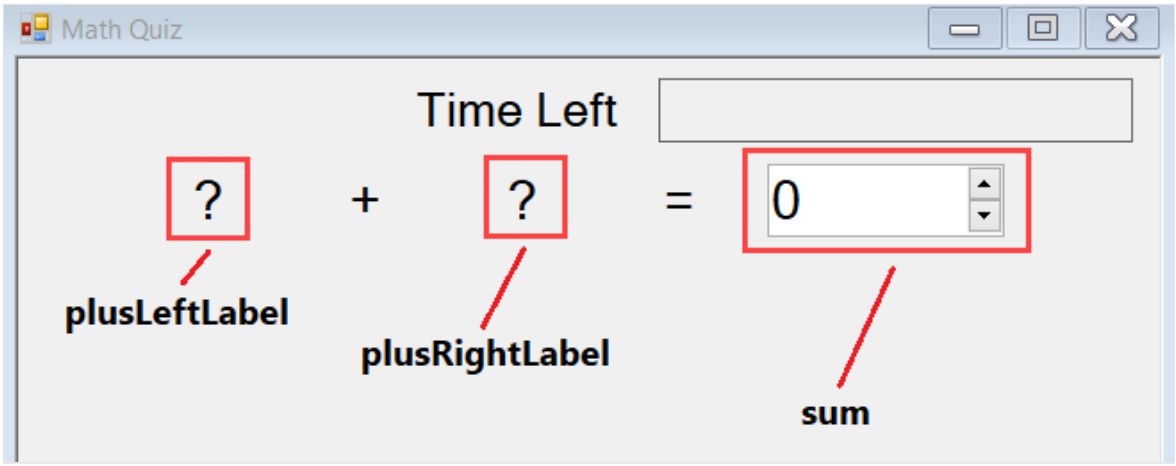
7. 移动该标签，使之整齐排列在“timeLabel”标签的左侧。



添加加法题的控件

测验的第一部分是加法题。 本部分介绍如何使用标签显示该题。

1. 从“工具箱”向窗体中添加“标签”控件。
2. 在“属性”框中，为标签设置以下属性：
   - 将“文本”设置为“?”（问号）。
   - 将“AutoSize”设置为“False”。
   - 将“Size”设置为“60, 50”。
   - 将字号设置为“18”。
   - 将“TextAlign”设置为“MiddleCenter”。
   - 将“Location”设置为“50, 75”，以便将此控件置于窗体上。
   - 将“(Name)”设置为“plusLeftLabel”。
3. 在窗体中，选择你创建的“plusLeftLabel”标签。 通过选择“编辑”>“复制”或按 Ctrl+C 来复制标签。
4. 通过选择“编辑”>“粘贴”或按 Ctrl+V 三次来将标签粘贴到窗体中三次。
5. 排列这三个新标签，使之在“plusLeftLabel”标签的右侧排成一行。
6. 将第二个标签的“文本”属性设置为 +（加号）。
7. 将第三个标签的“(Name)”属性设置为“plusRightLabel”。
8. 将第四个标签的“文本”属性设置为 =（等于号）。
9. 从“工具箱”向窗体中添加 NumericUpDown 控件。 稍后您将了解到有关此类控件的详细信息。
10. 在“属性”框中，为 NumericUpDown 控件设置以下属性：
    - 将字号设置为“18”。
    - 将宽度设置为 100。
    - 将“(Name)”设置为 sum。
11. 将此“NumericUpDown”控件与加法题的各 Label 控件排成一行。



添加减法、乘法和除法题的控件

接下来，向窗体中添加其他数学题的标签。

1. 复制 4 个标签控件以及你为加法问题创建的 NumericUpDown 控件。 将它们粘贴到窗体中。

2. 将新控件移动到加法控件下方并对齐。

3. 在“属性”框中，为新控件设置以下属性：

   - 将第一个问号标签的“(Name)”设置为“minusLeftLabel” 。
   - 将第二个标签的“文本”设置为 -（减号）。
   - 将第二个问号标签的“(Name)”设置为“minusRightLabel” 。
   - 将“NumericUpDown”控件的“(Name)”设置为“difference” 。

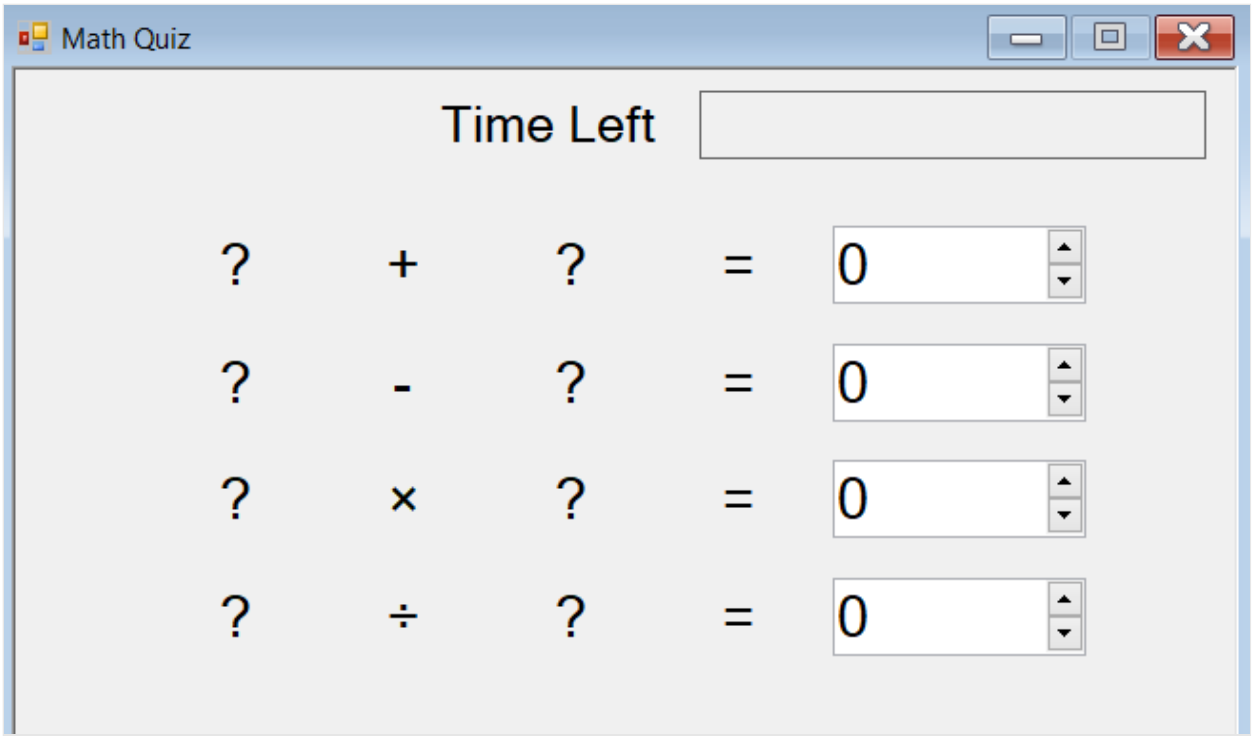
4. 复制加法控件，并将其粘贴到窗体中两次。

5. 对于第三行：

   - 将第一个问号标签的“(Name)”设置为“timesLeftLabel” 。
   - 将第二个标签的“Text”设置为 ×（乘号） 。 你可以复制本教程中的乘号，并将其粘贴到窗体中。
   - 将第二个问号标签的“(Name)”设置为“timesRightLabel” 。
   - 将“NumericUpDown”控件的“(Name)”设置为“product” 。

6. 对于第四行：

   - 将第一个问号标签的“(Name)”设置为“dividedLeftLabel” 。
   - 将第二个标签的“Text”设置为 ÷（除号） 。 你可以复制本教程中的除号，并将其粘贴到窗体中。
   - 将第二个问号标签的“(Name)”设置为“dividedRightLabel” 。
   - 将“NumericUpDown”控件的“(Name)”设置为“quotient” 。



添加“开始”按钮并设置 Tab 键索引顺序

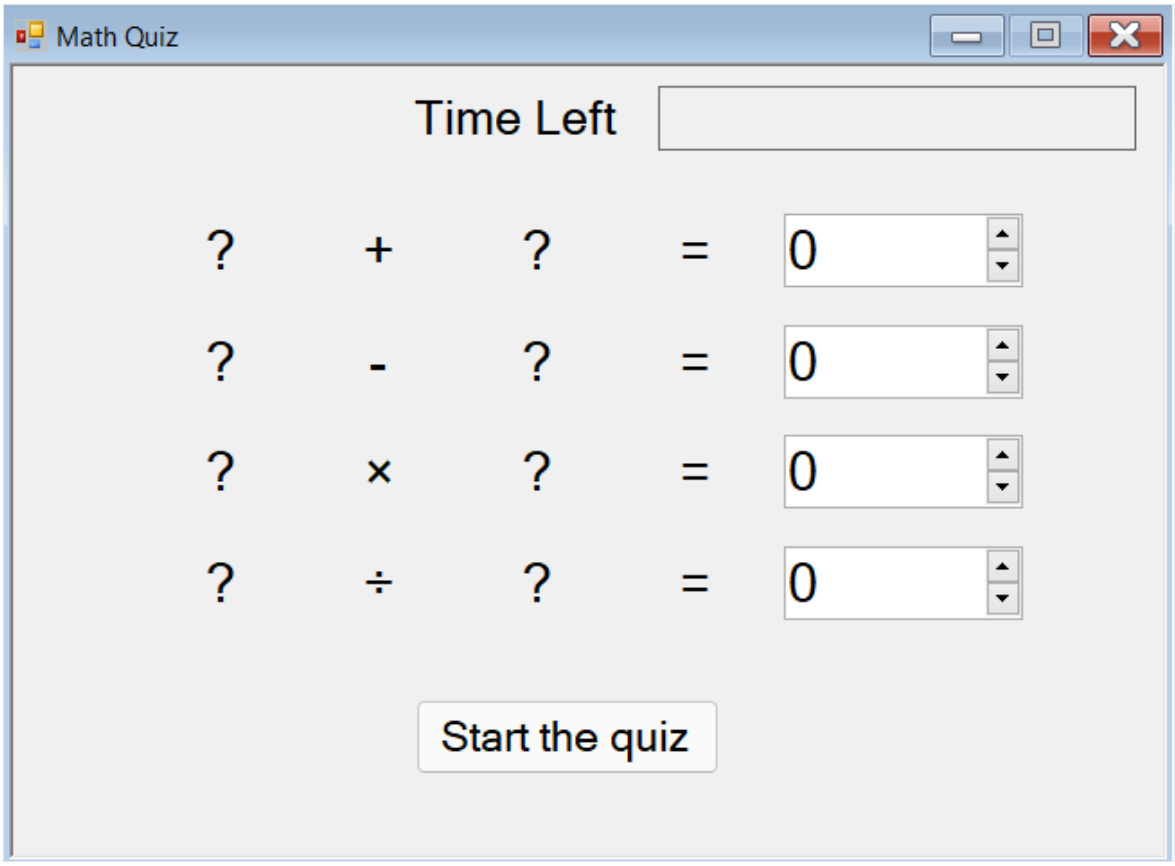
本部分介绍如何添加“开始”按钮。 你还将指定控件的 Tab 键顺序。 此顺序决定测验者使用 Tab 键从一个控件移动到下一个控件的顺序。

1. 从“工具箱”向窗体中添加 Button 控件。

2. 在“属性”框中，为按钮设置下列属性：

   - 将“(Name)”设置为“startButton” 。
   - 将“Text”设置为“开始测验” 。
   - 将字号设置为“14”。
   - 将“AutoSize”设置为“True”，这可使此按钮自动调整大小以适合文本 。
   - 将“TabIndex”设置为“0” 。 此值可使“开始”按钮成为接收焦点的第一个控件。

3. 将此按钮置于窗体底部附近的中心位置。



4. 在“属性”框中，设置每个 NumericUpDown 控件的“TabIndex”属性：

   - 将 sum NumericUpDown 控件的“TabIndex”设置为“1”。
   - 将“difference”NumericUpDown 控件的“TabIndex”设置为“2”。
   - 将“product”NumericUpDown 控件的“TabIndex”设置为“3”。
   - 将“quotient”NumericUpDown 控件的“TabIndex”设置为“4”。

运行应用

数学题尚不适用于你的测验。 但你仍可以运行应用来检查 TabIndex 值是否按预期方式工作。

1. 使用以下其中一种方法保存应用：

   - 按 Ctrl+Shift+S。
   - 在菜单栏中，选择“文件”>“保存全部”。
   - 在工具栏上，选择“保存全部”按钮。

2. 使用下列方法之一运行应用：

   - 按 F5。
   - 在菜单栏上，依次选择“调试”>“开始调试”。
   - 在工具栏上，选择“开始”按钮。

3. 按 Tab 键几次，以查看焦点如何从一个控件移动到下一个控件。

后续步骤

前往下一教程，为数学测验添加随机数学题和事件处理程序。

教程第2部分：向数学测验 WinForms 应用添加数学题

教程：向数学测验 WinForms 应用中添加数学题

项目 • 2024/01/13

在本系列的四个教程中，你将创建一个数学测验。 测验包含四个随机数学问题，测验者需要尝试在指定的时间内回答这些问题。

控件使用 C# 或 Visual Basic 代码。 在此第二个教程中，你将添加基于随机数的数学题的代码，从而让测验变得有挑战性。 还会创建一个名为 `StartTheQuiz()` 的方法来填充问题。

在第二个教程中，你将了解如何：

- ✔ 编写代码以创建用于数学题的 Random 对象。
- ✔ 为开始按钮添加事件处理程序。
- ✔ 编写代码以开始测验。

先决条件

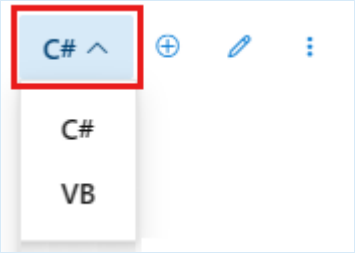
本教程基于前面的教程创建数学测验 WinForms 应用构建。 如果你尚未完成该教程，请先完成该教程。

创建随机加法问题

1. 在你的 Visual Studio 项目中，选择“Windows 窗体设计器”。
2. 选择窗体“Form1”。
3. 在菜单栏上，选择“视图”>“代码” 。 此时将显示 Form1.cs 或 Form1.vb（具体取决于你所使用的编程语言），以便你查看窗体背后的代码。
4. 通过在代码顶部附近添加 `new` 语句来创建一个 Random 对象。

C#

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();
```

> **ⓘ 重要**
>
> 使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。
>
> 
>

可以使用与此类似的 `new` 语句创建按钮、标签、面板、OpenFileDialogs、ColorDialogs、SoundPlayers、Randoms，甚至是窗体。 这些项称为“对象”。

运行程序时，窗体将启动。 窗体背后的代码将创建一个 Random 对象并将其命名为“randomizer”。

你的测验需要变量来存储其为每个问题创建的随机数字。 使用变量前，请声明这些变量，也就是列出它们的名称和数据类型。

1. 将两个整型变量添加到窗体，并将它们分别命名为“addend1”和“addend2”。

   > **ⓘ 备注**
   >
   > 整数变量在 C# 中表示为“int”，在 Visual Basic 中表示为“Integer” 。 这种变量可以存储从 -2147483648 到 2147483647 的正负数，并仅能存储整数而不能存储小数。

   应使用与添加 Random 对象相似的语法来添加整数变量，如下面的代码所示。

   C#

   ```
   // Create a Random object called randomizer
   // to generate random numbers.
   Random randomizer = new Random();

   // These integer variables store the numbers
   // for the addition problem.
   int addend1;
   int addend2;
   ```

1. 添加名为 `StartTheQuiz()` 的方法。 此方法使用 Random 对象的 Next() 方法来为标签生成随机数字。 `StartTheQuiz()` 最终将填充所有问题，然后启动计时器，因此请将此信息添加到摘要注释中。 该函数应类似于以下代码。

C#

```
/// <summary>
/// Start the quiz by filling in all of the problems
/// and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;
}
```

对 Random 对象使用 `Next()` 方法时（例如，在调用 `randomizer.Next(51)` 时），将获得一个小于 51（或介于 0 和 50）的随机数。 此代码将调用 `randomizer.Next(51)`，以便两个随机数相加的答案介于 0 和 100。

详细了解这些语句。

C#

```
plusLeftLabel.Text = addend1.ToString();
plusRightLabel.Text = addend2.ToString();
```

这些语句设置“plusLeftLabel”和“plusRightLabel”的“Text”属性，以便它们显示两个随机数。 标签控件以文本格式显示值，并且在编程中，字符串保存文本。 每个整数的

`ToString()` 方法都会将整数转换为标签可以显示的文本。

创建随机减法、乘法和除法题

下一步是声明变量并为其他数学题提供随机值。

1. 在加法题变量后，向窗体中添加其余数学题的整数变量。 代码应类似于以下示例。

C#

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();

    // These integer variables store the numbers
    // for the addition problem.
    int addend1;
    int addend2;

    // These integer variables store the numbers
    // for the subtraction problem.
    int minuend;
    int subtrahend;

    // These integer variables store the numbers
    // for the multiplication problem.
    int multiplicand;
    int multiplier;

    // These integer variables store the numbers
    // for the division problem.
    int dividend;
    int divisor;
```

---

1. 通过添加以下代码来修改 `StartTheQuiz()` 方法，以“Fill in the subtraction problem”注释开头。

C#

```
/// <summary>
/// Start the quiz by filling in all of the problem
/// values and starting the timer.
```

```
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;

    // Fill in the subtraction problem.
    minuend = randomizer.Next(1, 101);
    subtrahend = randomizer.Next(1, minuend);
    minusLeftLabel.Text = minuend.ToString();
    minusRightLabel.Text = subtrahend.ToString();
    difference.Value = 0;

    // Fill in the multiplication problem.
    multiplicand = randomizer.Next(2, 11);
    multiplier = randomizer.Next(2, 11);
    timesLeftLabel.Text = multiplicand.ToString();
    timesRightLabel.Text = multiplier.ToString();
    product.Value = 0;

    // Fill in the division problem.
    divisor = randomizer.Next(2, 11);
    int temporaryQuotient = randomizer.Next(2, 11);
    dividend = divisor * temporaryQuotient;
    dividedLeftLabel.Text = dividend.ToString();
    dividedRightLabel.Text = divisor.ToString();
    quotient.Value = 0;
```

此代码在 Random 类的 Next() 方法的使用方式上与加法题略有不同。 当您为 `Next()` 方法赋予两个值时，它会选取一个大于等于第一个值但小于第二个值的随机数。

通过将 `Next()` 方法与两个参数一起使用，可以确保减法题的答案是正数，乘法题答案的最大值为 100，而除法题的答案不是分数。

为开始按钮添加事件处理程序

在本部分中，你将添加代码，以便在选择“开始”按钮时开始测验。 为响应按钮选择等事件而运行的代码称为事件处理程序。

1. 在“Windows 窗体设计器”中，双击“开始测验”按钮，或选择该按钮并按 Enter 键 。 此时将显示窗体的代码，并显示新方法。

   这些操作将向“开始”按钮添加一个 Click 事件处理程序。 当测验者选择此按钮时，应用将运行你要添加到此新方法中的代码。

2. 添加以下两个语句，以使事件处理程序开始测验。

   C#

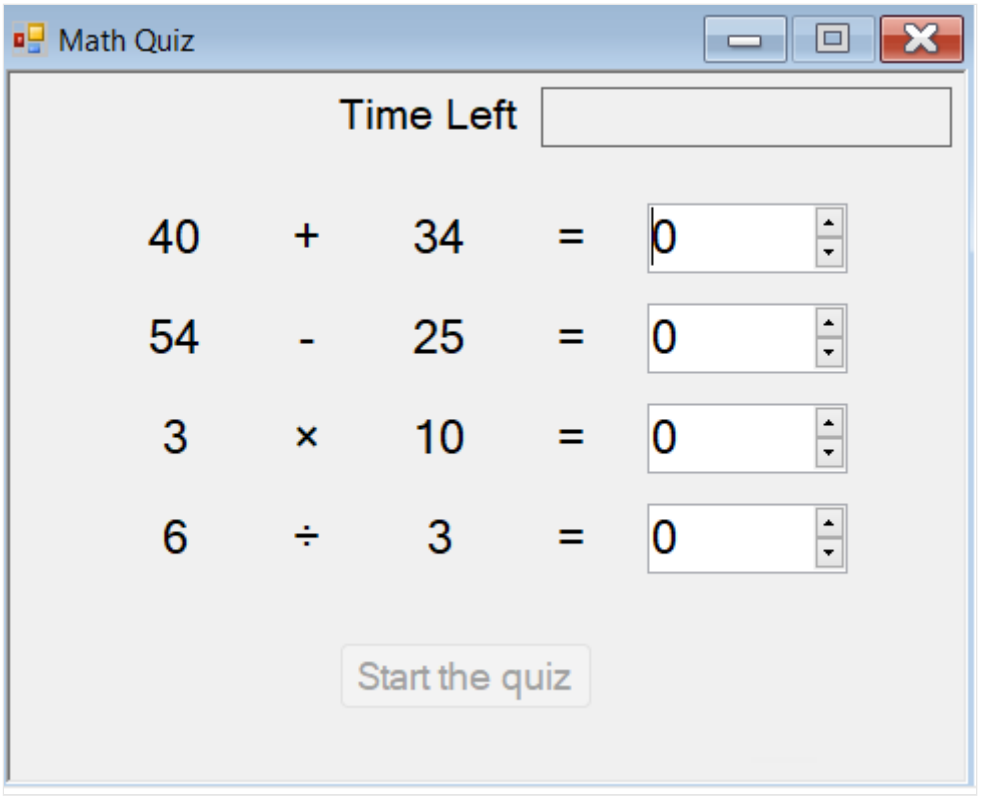
   ```
   private void startButton_Click(object sender, EventArgs e)
   {
       StartTheQuiz();
       startButton.Enabled = false;
   }
   ```

第一个语句将调用新的 `StartTheQuiz()` 方法。 第二个语句将“startButton”控件的“Enabled”属性设置为 `false`，从而使测验者在测验期间不能选择此按钮 。

运行应用

1. 保存代码。

2. 运行应用，然后选择“开始测验”。 随即出现随机数学题，如以下屏幕截图所示。



后续步骤

前往下一教程，将计时器添加到数学测验中并检查用户答案。

教程第 3 部分：向数学测验 WinForms 应用添加计时器控件

反馈

此页面是否有帮助? 是 否

提供产品反馈 | 询问社区

教程：向数学测验 WinForms 应用添加计时器

项目 • 2023/03/21

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本系列的四个教程中，你将创建一个数学测验。 测验包含四个随机数学问题，测验者需要尝试在指定的时间内回答这些问题。

测验使用计时器控件。 此控件背后的代码跟踪运行时间，并检查测验者的答案。

在第三个教程中，你将了解如何：

- ✓ 添加计时器控件。
- ✓ 为计时器添加事件处理程序。
- ✓ 编写代码以检查用户的答案、显示消息并填写正确的答案。

先决条件

本教程基于前面的教程（从创建数学测验 WinForms 应用开始）构建。 如果尚未完成这些教程，请先完成这些教程。

添加倒计时计时器

若要在测验期间持续跟踪时间，请使用计时器组件。 还需要一个变量来存储剩余的时间量。

1. 添加名为“timeLeft”的整数变量，其方式与在之前的教程中声明变量的方式相同。 将“timeLeft”声明放在紧靠在其他声明之后的位置。 代码应类似于以下示例。

C#

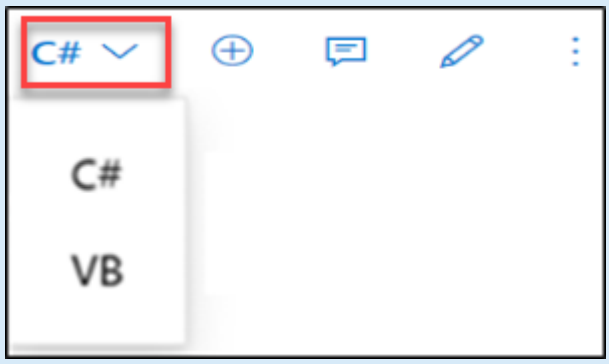
C#

```
public partial class Form1 : Form
{
    // Create a Random object called randomizer
    // to generate random numbers.
    Random randomizer = new Random();

    // These integer variables store the numbers
    // for the addition problem.
    int addend1;
```

```
    int addend2;

    // These integer variables store the numbers
    // for the subtraction problem.
    int minuend;
    int subtrahend;

    // These integer variables store the numbers
    // for the multiplication problem.
    int multiplicand;
    int multiplier;

    // These integer variables store the numbers
    // for the division problem.
    int dividend;
    int divisor;

    // This integer variable keeps track of the
    // remaining time.
    int timeLeft;
```

ⓘ 重要

使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



2. 在“Windows 窗体设计器”中，将“工具箱”的“组件”类别中的 Timer 控件移到窗体中。 此控件显示在设计窗口底部的灰色区域内。

3. 在窗体中，请选择刚才添加的“timer1”图标，并将其“Interval”属性设置为“1000”。 由于此间隔以毫秒为单位，因此值为 1000 时会导致计时器每秒引发一个 Tick 事件。

检查答案

由于计时器每秒引发一个 Tick 事件，所以在 Tick 事件处理程序中应该检查运行时间。 在该事件处理程序中检查答案也是可行的。 如果时间已用完，或者答案正确，则测验应结束。

在编写该事件处理程序之前，请添加一个调用 `CheckTheAnswer()` 的方法，用于确定数学题的答案是否正确。 此方法应与其他方法（如 `StartTheQuiz()`）保持一致。 代码应类似于以下示例。

C#

C#

```
/// <summary>
/// Check the answers to see if the user got everything right.
/// </summary>
/// <returns>True if the answer's correct, false otherwise.</returns>
private bool CheckTheAnswer()
{
    if ((addend1 + addend2 == sum.Value)
        && (minuend - subtrahend == difference.Value)
        && (multiplicand * multiplier == product.Value)
        && (dividend / divisor == quotient.Value))
        return true;
    else
        return false;
}
```

此方法确定数学题的答案，并将结果与 NumericUpDown 控件中的值进行比较。 在此代码中：

- Visual Basic 版本使用 `Function` 关键字而不是一般的 `Sub` 关键字，因为此方法将返回一个值。
- 由于使用键盘无法方便地输入乘号 (×) 和除号 (÷)，因此 C# 和 Visual Basic 接受用星号 (*) 代替乘号，用斜线 (/) 代替除号。
- 在 C# 中，`&&` 是 `logical and` 运算符。 在 Visual Basic 中，等效的运算符是 `AndAlso`。 使用 `logical and` 运算符检查多个条件是否为 true。 在这种情况下，如果值都正确，则方法返回值 `true`。 否则，此方法将返回值 `false`。
- `if` 语句使用 NumericUpDown 控件的 Value 属性访问控件的当前值。 下一部分使用相同的属性在每个控件中显示正确答案。

向计时器添加事件处理程序

现已具有检查答案的方法，接下来可以编写 Tick 事件处理程序的代码。 在计时器引发 Tick 事件后，此代码每秒运行一次。 此事件处理程序通过调用 `CheckTheAnswer()` 来检查测验者的答案。 它还检查测验已消耗的时间。

1. 在窗体中，双击“计时器”控件，或将其选中，然后选择 Enter 键 。 这些操作将向计时器添加 Tick 事件处理程序。 代码编辑器将出现并显示 Tick 处理程序的方法。

2. 将下面的语句添加到新事件处理程序方法。

C#

C#

```
private void timer1_Tick(object sender, EventArgs e)
{
    if (CheckTheAnswer())
    {
        // If CheckTheAnswer() returns true, then the user
        // got the answer right. Stop the timer
        // and show a MessageBox.
        timer1.Stop();
        MessageBox.Show("You got all the answers right!",
                        "Congratulations!");
        startButton.Enabled = true;
    }
    else if (timeLeft > 0)
    {
        // If CheckTheAnswer() returns false, keep counting
        // down. Decrease the time left by one second and
        // display the new time left by updating the
        // Time Left label.
        timeLeft = timeLeft - 1;
        timeLabel.Text = timeLeft + " seconds";
    }
    else
    {
        // If the user ran out of time, stop the timer, show
        // a MessageBox, and fill in the answers.
        timer1.Stop();
        timeLabel.Text = "Time's up!";
        MessageBox.Show("You didn't finish in time.", "Sorry!");
        sum.Value = addend1 + addend2;
        difference.Value = minuend - subtrahend;
        product.Value = multiplicand * multiplier;
        quotient.Value = dividend / divisor;
        startButton.Enabled = true;
    }
}
```

在测验中的每一秒，此方法都将运行。 代码首先检查 `CheckTheAnswer()` 返回的值。

- 如果所有答案都正确，该值为 `true`，且测验结束：
  - 计时器停止。
  - 随即出现一条祝贺消息。
  - startButton 控件的“Enabled”属性将设置为 `true`，以便测验者可以开始其他测试。
- 如果 `CheckTheAnswer()` 返回 `false`，代码将检查 timeLeft 的值：
  - 如果此变量大于 0，计时器将从 timeLeft 中减去 1。 它还将更新“timeLabel”控件的“Text”属性，以便向测验者显示剩余的秒数 。
  - 如果没有剩余时间，计时器将停止并更改 timeLabel 控件的文本，使之显示“时间到!”一个消息框宣布测验结束，并显示答案。 “开始”按钮将再次可用。

启动计时器

若要在测验开始时启动计时器，请向 `StartTheQuiz()` 方法的末尾添加三行，如以下示例所示。

C#

C#

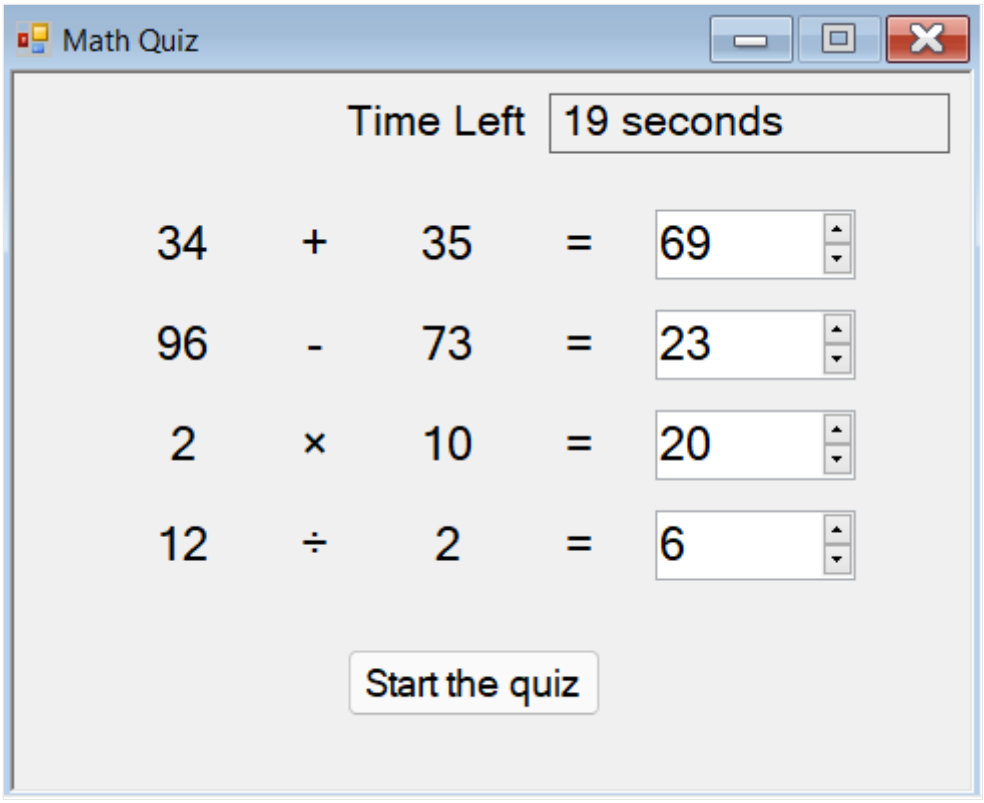
```
/// <summary>
/// Start the quiz by filling in all of the problem
/// values and starting the timer.
/// </summary>
public void StartTheQuiz()
{
    // Fill in the addition problem.
    // Generate two random numbers to add.
    // Store the values in the variables 'addend1' and 'addend2'.
    addend1 = randomizer.Next(51);
    addend2 = randomizer.Next(51);

    // Convert the two randomly generated numbers
    // into strings so that they can be displayed
    // in the label controls.
    plusLeftLabel.Text = addend1.ToString();
    plusRightLabel.Text = addend2.ToString();

    // 'sum' is the name of the NumericUpDown control.
    // This step makes sure its value is zero before
    // adding any values to it.
    sum.Value = 0;

    // Fill in the subtraction problem.
```

```
    minuend = randomizer.Next(1, 101);
    subtrahend = randomizer.Next(1, minuend);
    minusLeftLabel.Text = minuend.ToString();
    minusRightLabel.Text = subtrahend.ToString();
    difference.Value = 0;

    // Fill in the multiplication problem.
    multiplicand = randomizer.Next(2, 11);
    multiplier = randomizer.Next(2, 11);
    timesLeftLabel.Text = multiplicand.ToString();
    timesRightLabel.Text = multiplier.ToString();
    product.Value = 0;

    // Fill in the division problem.
    divisor = randomizer.Next(2, 11);
    int temporaryQuotient = randomizer.Next(2, 11);
    dividend = divisor * temporaryQuotient;
    dividedLeftLabel.Text = dividend.ToString();
    dividedRightLabel.Text = divisor.ToString();
    quotient.Value = 0;

    // Start the timer.
    timeLeft = 30;
    timeLabel.Text = "30 seconds";
    timer1.Start();
}
```

当测验开始时，此代码将“timeLeft”变量将设置为 30，而 timeLabel 控件的“Text”属性将为 30 秒 。 然后，Timer 控件的 Start() 方法将开始倒计时。

运行应用

1. 保存并运行程序。
2. 选择“开始测验”。 计时器将开始倒计时。 当时间用完时，测验将结束，并显示答案。
3. 开始另一个测验，并提供数学题的正确答案。 在时间限制内回答正确时，将出现一个消息框、开始按钮变为可用并且计时器停止。



后续步骤

请继续学习下一教程，了解如何自定义数学测验。

教程第 4 部分：自定义数学测验 WinForms 应用

教程：自定义数学测验 WinForms 应用

项目 • 2023/04/20

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本系列的四个教程中，你将创建一个数学测验。 测验包含四个随机数学问题，测验者需要尝试在指定的时间内回答这些问题。

本教程介绍如何通过清除默认值和自定义控件的外观来强化测验。

在此最后一个教程中，你将了解如何：

- ✓ 添加事件处理程序以清除默认的 NumericUpDown 控件值。
- ✓ 自定义测验。

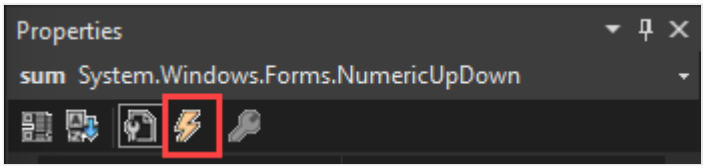
先决条件

本教程基于前面的教程（从创建数学测验 WinForms 应用开始）构建。 如果尚未完成这些教程，请先完成这些教程。

为 NumericUpDown 控件添加事件处理程序

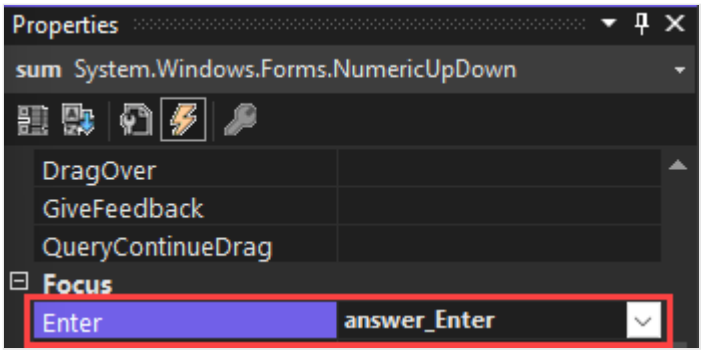
测验包含测验者用于输入数字的 NumericUpDown 控件。 输入答案时，需要先选择默认值，或手动删除该值。 通过添加 Enter 事件处理程序，可以更轻松地输入答案。 当测验对象选择每个 NumericUpDown 控件中的当前值并开始输入其他值时，此代码将立即选中并清除该当前值。

1. 选择窗体上的第一个 NumericUpDown 控件。 在“属性”对话框中，选择工具栏上的“事件”图标 。

   “属性”中的“事件”选项卡显示窗体中所选项的所有可响应的事件 。 在这种情况下，列出的所有事件都与 NumericUpDown 控件相关。

2. 选择“Enter”事件，输入“answer_Enter”，然后按 Enter 键 。



代码编辑器将出现并显示你为 sum NumericUpDown 控件创建的 Enter 事件处理程序。

3. 在“answer_Enter”事件处理程序的方法中，添加以下代码：

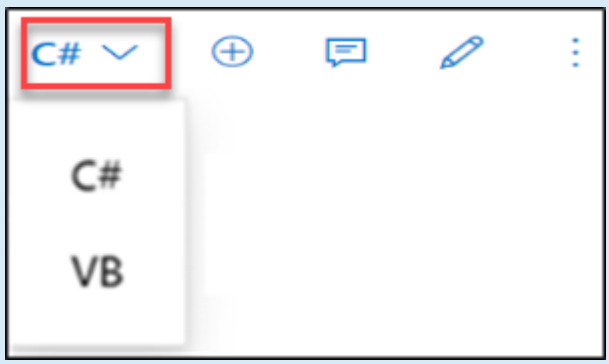
C#

C#

```
private void answer_Enter(object sender, EventArgs e)
{
    // Select the whole answer in the NumericUpDown control.
    NumericUpDown answerBox = sender as NumericUpDown;

    if (answerBox != null)
    {
        int lengthOfAnswer = answerBox.Value.ToString().Length;
        answerBox.Select(0, lengthOfAnswer);
    }
}
```

ⓘ 重要

使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



在此代码中：

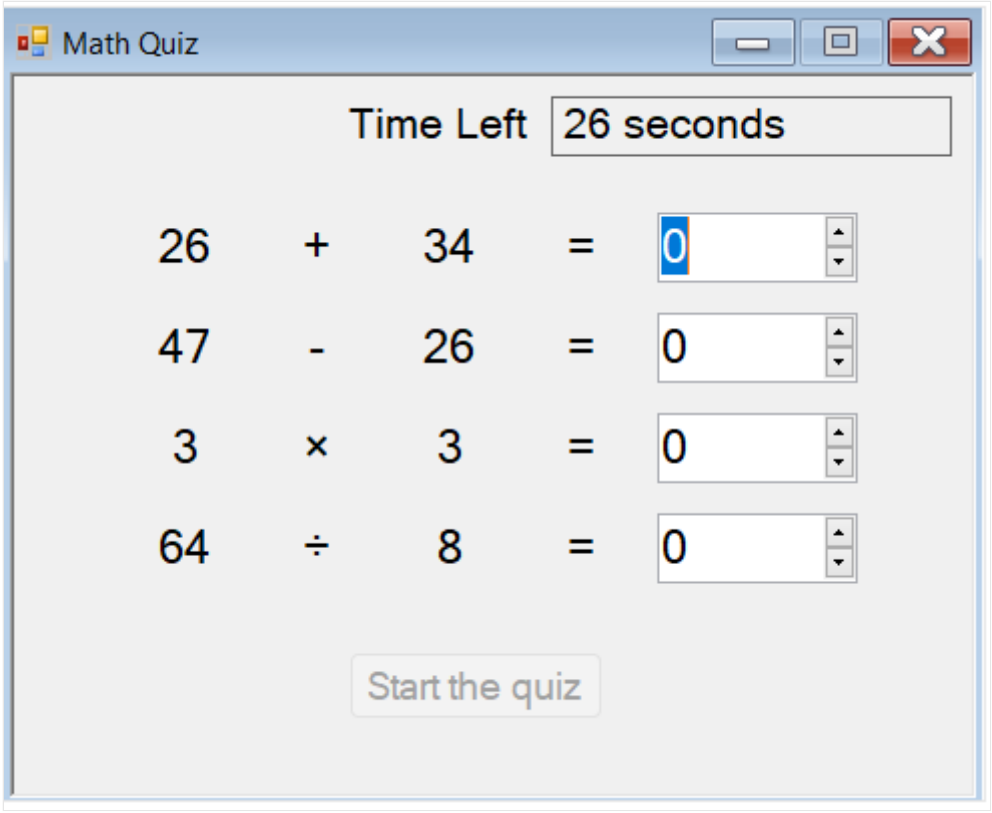
- 第一行声明该方法。 它包含名为 `sender` 的参数。 在 C# 中，该参数为 `object sender`。 在 Visual Basic 中，它为 `sender As System.Object`。 此参数引用已触发其事件的对象，即发送方。 在此示例中，发送方对象为 NumericUpDown 控件。
- 方法中的第一行将发送方从泛型对象强制转换或转换为 NumericUpDown 控件。 该行还将名称“answerBox”分配给 NumericUpDown 控件。 窗体上的所有 NumericUpDown 控件都将使用此方法，而不只是加法问题的控件。
- 下一行验证 answerBox 是否已成功地强制转换为一个 NumericUpDown 控件。
- `if` 语句中的第一行确定 NumericUpDown 控件中当前答案的长度。
- `if` 语句中的第二行使用答案长度来选择控件中的当前值。

当测验者选择此控件时，Visual Studio 将触发此事件。 此代码将选择当前答案。 一旦测验者开始输入其他答案，当前的答案将被清除并替换为新答案。

1. 在“Windows 窗体设计器”中，再次选择加法问题的 NumericUpDown 控件。
2. 在“属性”对话框的“事件”页中，找到“Click”事件，然后从下拉列表中选择“answer_Enter”。 这是你刚添加的事件处理程序。
3. 在“Windows 窗体设计器”中，选择减法问题的 NumericUpDown 控件。
4. 在“属性”对话框的“事件”页中，找到“Enter”事件，然后从下拉列表中选择“answer_Enter” 。 这是你刚添加的事件处理程序。 对“Click”事件重复此步骤。
5. 对乘法和除法 NumericUpDown 控件重复上述两个步骤。

运行应用

1. 保存并运行程序。
2. 开始测验，然后选择 NumericUpDown 控件。 系统将自动选中现有值，然后在你开始输入其他值时自动清除现有值。



自定义测验

在本教程的最后一部分中，你将了解一些自定义测验和扩展所学内容的方式。

更改标签的颜色

- 使用“timeLabel”控件的“BackColor”属性，使此标签在测验只剩下 5 秒时变为红色 。

C#

C#

```
timeLabel.BackColor = Color.Red;
```

- 当测验结束时重置颜色。

为正确答案播放声音

当测验参加者在 NumericUpDown 控件中输入正确答案时，通过播放声音来进行提示。 为实现此功能，请为每个控件的 ValueChanged 事件编写事件处理程序。 每当测验者更改控件的值时，此类事件便会触发。

后续步骤

祝贺！ 你已完成本系列教程。 你已在 Visual Studio IDE 中完成以下编程和设计任务：

- 创建了一个使用 Windows 窗体的 Visual Studio 项目
- 添加标签、按钮和 NumericUpDown 控件
- 添加计时器
- 为控件设置事件处理程序
- 编写了 C# 或 Visual Basic 代码来处理事件

继续学习另一个关于如何构建匹配游戏的教程系列。

教程 3：创建匹配游戏

教程：创建匹配游戏 WinForms 应用

项目 • 2023/05/08

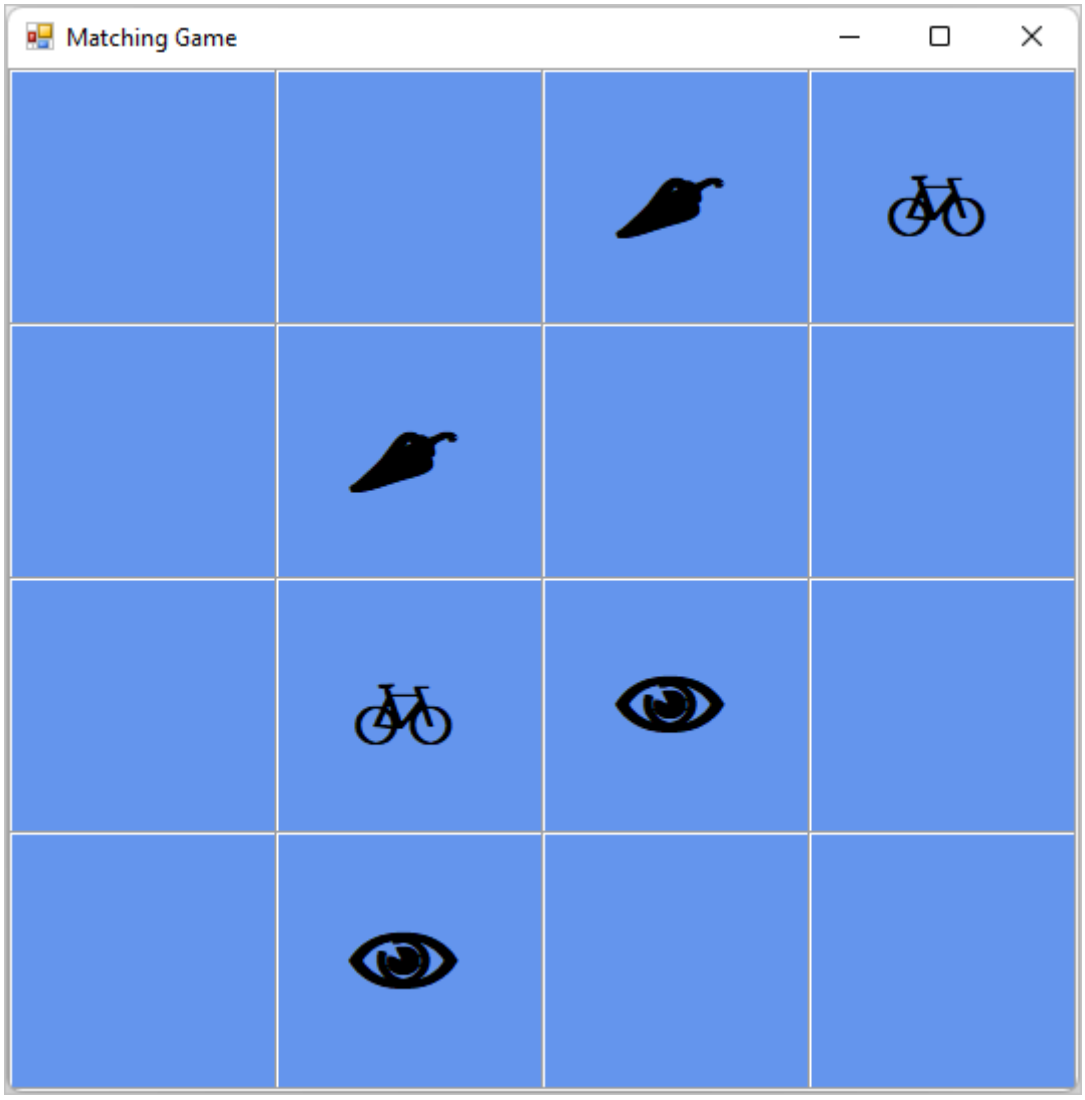
适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在这四个教程系列中，你将构建一个匹配游戏，玩家在其中匹配隐藏的图标对。

使用这些教程了解 Visual Studio 集成设计环境 (IDE) 中的以下任务。

- 在 List<T> 对象中存储对象，例如图标。
- 使用 `foreach` 循环（C# 中）或 `For Each` 循环（Visual Basic 中）循环访问列表中的各项。
- 使用引用变量跟踪窗体的状态。
- 生成事件处理程序，以响应可用于多个对象的事件。
- 创建一个计时器，进行倒计时，然后在启动后立即准确触发事件。

完成后，你将拥有一个完整的游戏。



在第一个教程中，你将了解如何：

- ✔ 创建一个使用 Windows 窗体的 Visual Studio 项目。
- ✔ 添加布局元素并设置其格式。
- ✔ 添加要显示的标签并设置其格式。

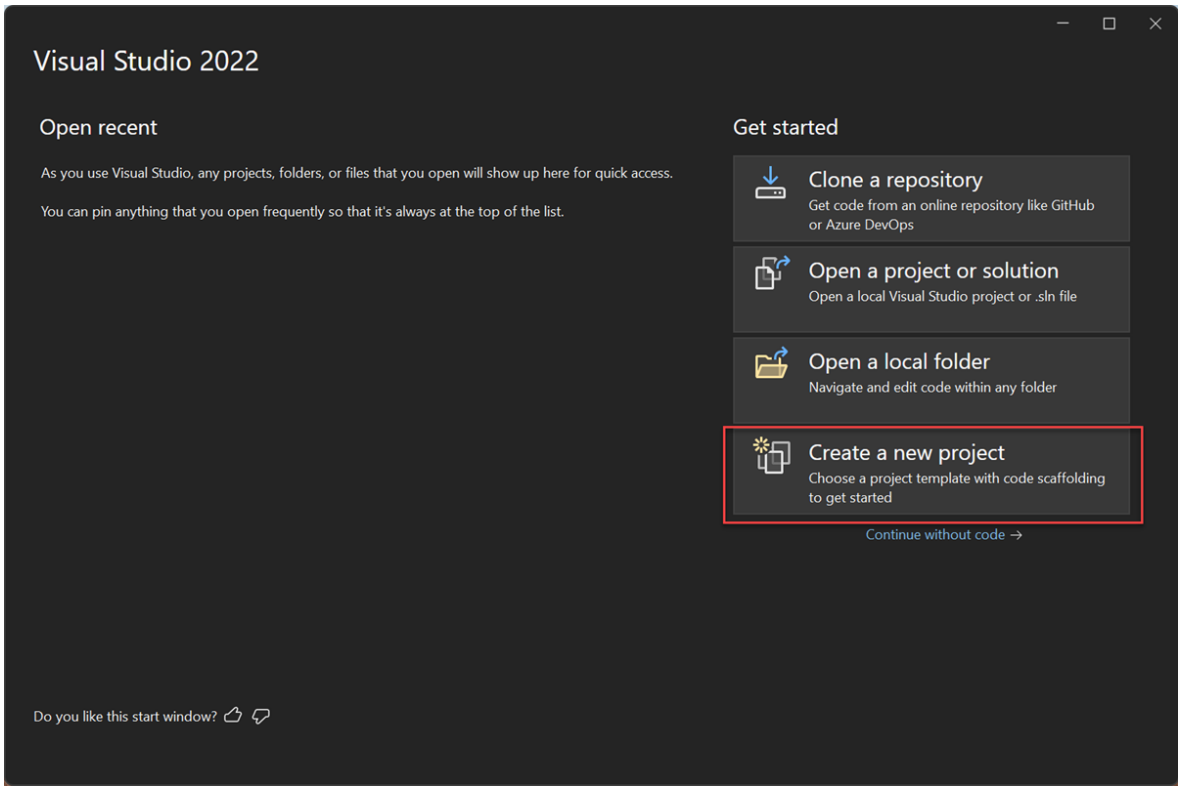
必备条件

若要完成本教程，必须具有 Visual Studio。 请访问 Visual Studio 下载页 获取免费版本。

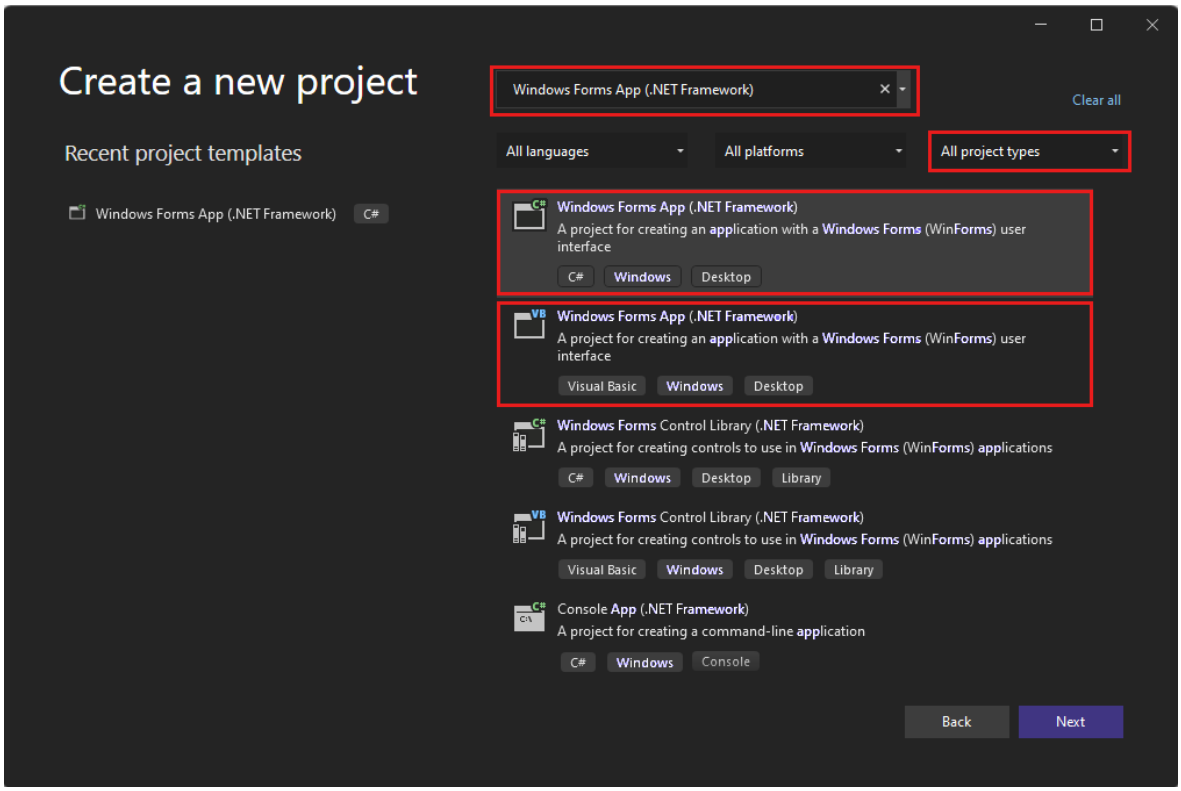
创建 Windows Forms 匹配游戏项目

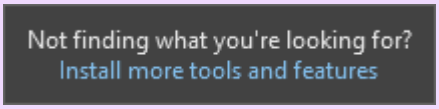
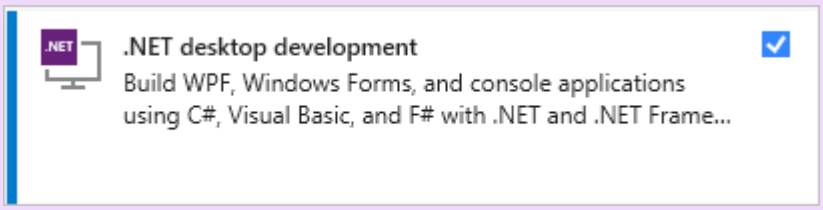
创建匹配游戏时，第一步是创建 Windows 窗体应用项目。

1. 打开 Visual Studio。
2. 在“开始”窗口中，选择“创建新项目”。

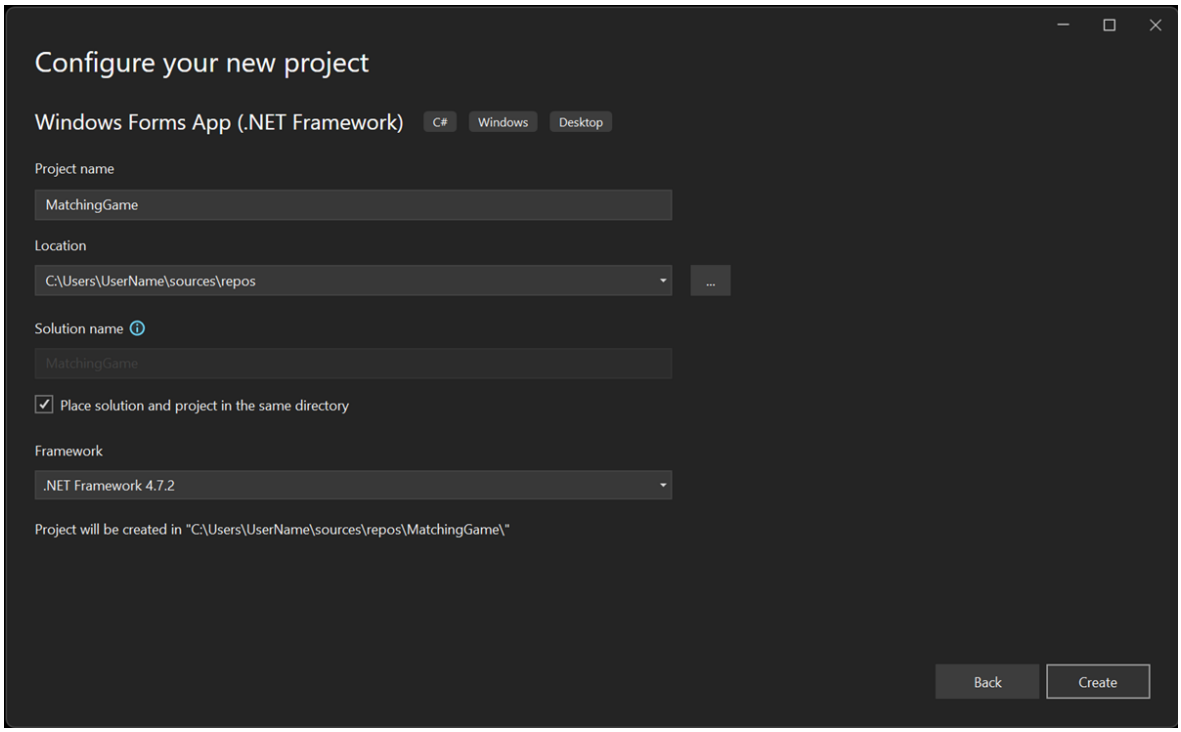


3. 在“创建新项目”窗口中，搜索“Windows 窗体”。 然后，从“所有项目类型”列表中选择“桌面” 。
4. 针对 C# 或 Visual Basic，选择“Windows 窗体应用(.NET Framework)”模板，然后选择“下一步” 。



> **① 备注**
>
> 如果未看到“Windows 窗体应用(.NET Framework)”模板，则可以通过“创建新项目”窗口安装该模板。 在“找不到所需内容?”消息中，选择“安装更多工具和功能”链接 。
>
> 
>
>
> 接下来，在 Visual Studio 安装程序中，选择“.NET 桌面开发”。
>
> 
>
>
> 在 Visual Studio 安装程序中，选择“修改”。 系统可能会提示你保存工作内容。 接下来，选择“继续”以安装工作负载。

5. 在“配置新项目”窗口中，将项目命名为“MatchingGame”，然后选择“创建”。



Visual Studio 将为你的应用创建解决方案。 解决方案是应用所需全部项目和文件的容器。

此时，Visual Studio 在 Windows 窗体设计器中显示一个空窗体。

为游戏创建布局

在本部分中，你将创建游戏的 4x4 网格。

1. 单击窗体以选择“Windows 窗体设计器”。 对于 C#，该选项卡显示 Form1.cs [Design]，对于 Visual Basic 则显示 Form1.vb [Design] 。 在“属性”窗口中，设置下列窗体属性。

    - 将“Text”属性从“Form1”更改为“Matching Game” 。 该文本显示在游戏窗口的顶部。
    - 设置窗体大小。 可以通过将“Size”属性设置为“550, 550”或拖动窗体的角直到在 Visual Studio IDE 底部看到正确的大小来更改它。

2. 选择 IDE 左侧的“工具箱”标签。 如果看不到它，请从菜单栏中选择“查看”>“工具箱”，或按 Ctrl+Alt+X” 。

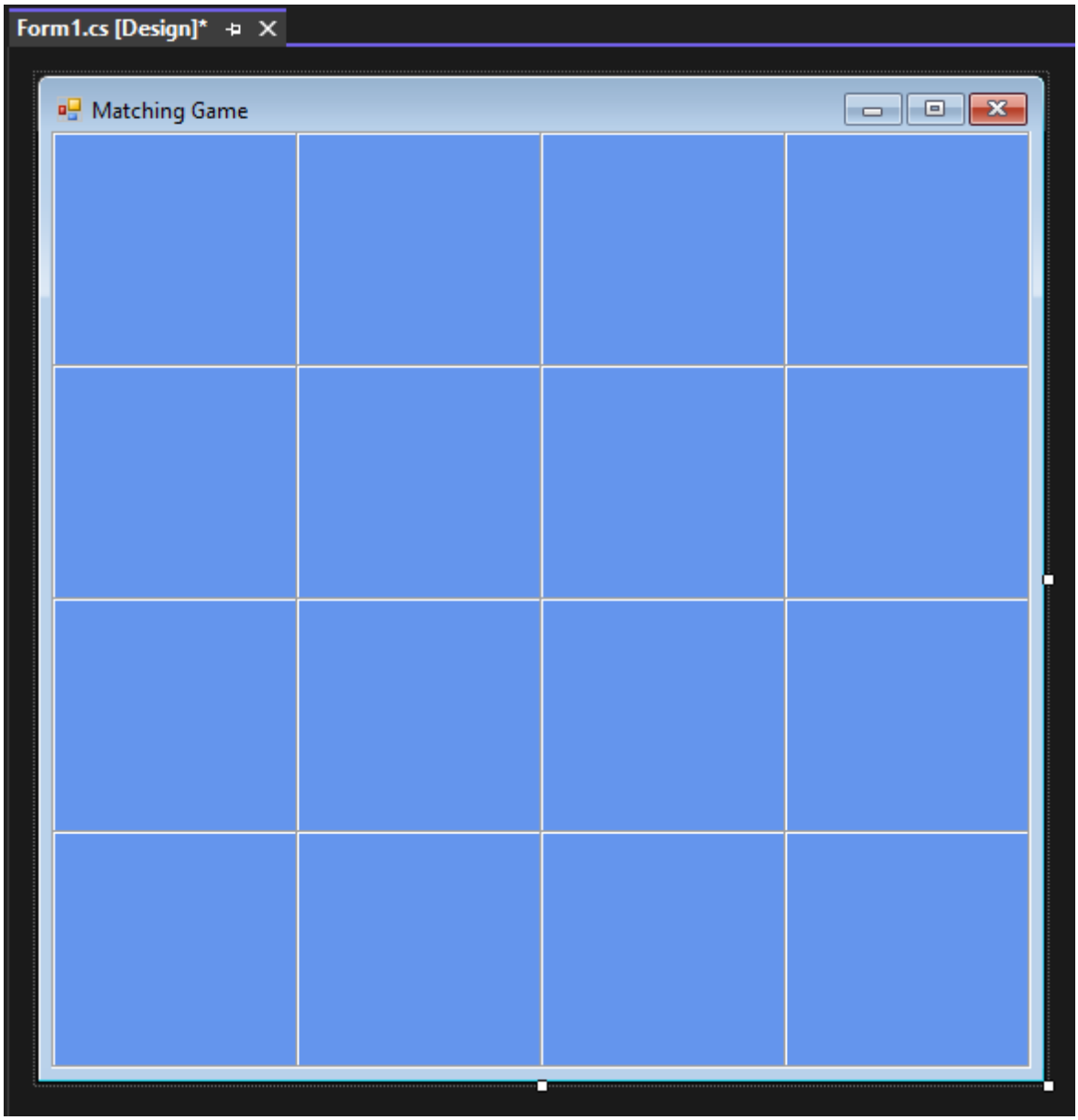
3. 从工具箱中的“容器”类别拖动 TableLayoutPanel 控件或双击它。 在“属性”窗口中为面板设置以下属性。

    - 将“BackColor”属性设置为“CornflowerBlue”。 若要设置此属性，请选择 BackColor 属性旁边的箭头。 在“BackColor”对话框中，选择“Web”。 在可用颜色名称中，选择“CornflowerBlue”。

> **ⓘ 备注**
>
> 这些颜色未按字母顺序排列，CornflowerBlue 位于列表底部附近。

- 通过选择大的中间按钮，从下拉列表中将“Dock”属性设置为“Fill”。 此选项会扩展该表，使其覆盖整个窗体。
- 将“CellBorderStyle”属性设置为“Inset”。 此值会在游戏板上每个单元格之间提供可视边框。
- 选择 TableLayoutPanel 右上角的三角形按钮，以显示任务菜单。 在任务菜单上，选择“添加行”两次以添加另外两行。 然后选择“添加列”两次以添加另外两列。
- 在任务菜单上，选择“编辑行和列”，打开“列和行样式”窗口 。 对于每一列，选择“百分比”选项，然后将每列的宽度设置为 25%。
- 从窗口顶部的列表中选择“行”，然后将每行的高度设置为 25%。
- 完成后，选择“确定”以保存更改。

TableLayoutPanel 现在是一个 4x4 网格，包含 16 个大小相等的方块单元格。 图标稍后会显示在这些行和列中。



添加要显示的标签并设置其格式

在本部分中，你将创建标签并设置其格式，以在游戏期间显示。

1. 确保在窗体编辑器中选择了该 TableLayoutPanel。 你应该会在“属性”窗口顶部看到“tableLayoutPanel1”。 如果未选中，请选择窗体上的 TableLayoutPanel，或从“属性”窗口顶部的列表中选择它。

2. 像以前一样打开工具箱，然后打开“公共控件”类别。 将 Label 控件添加到 TableLayoutPanel 的左上方单元格。 标签控件现已在 IDE 中选中。 为其设置下列属性。

    - 将标签的“BackColor”属性设置为“CornflowerBlue” 。
    - 将“AutoSize”属性设置为“False”。
    - 将“Dock”属性设置为“Fill”。

- 通过选择“TextAlign”属性旁的下拉按钮并选择中间按钮，将该属性设置为“MiddleCenter” 。 此值将确保图标显示在单元格中间。
- 选择“Font”属性。 此时会显示一个省略号 (...) 按钮。 选择省略号，并将“Font”值设置为“Webdings”，将“Font Style”设置为“Bold”，并将“Size”设置为“48” 。
- 将标签的“Text”属性设置为字母“c”。

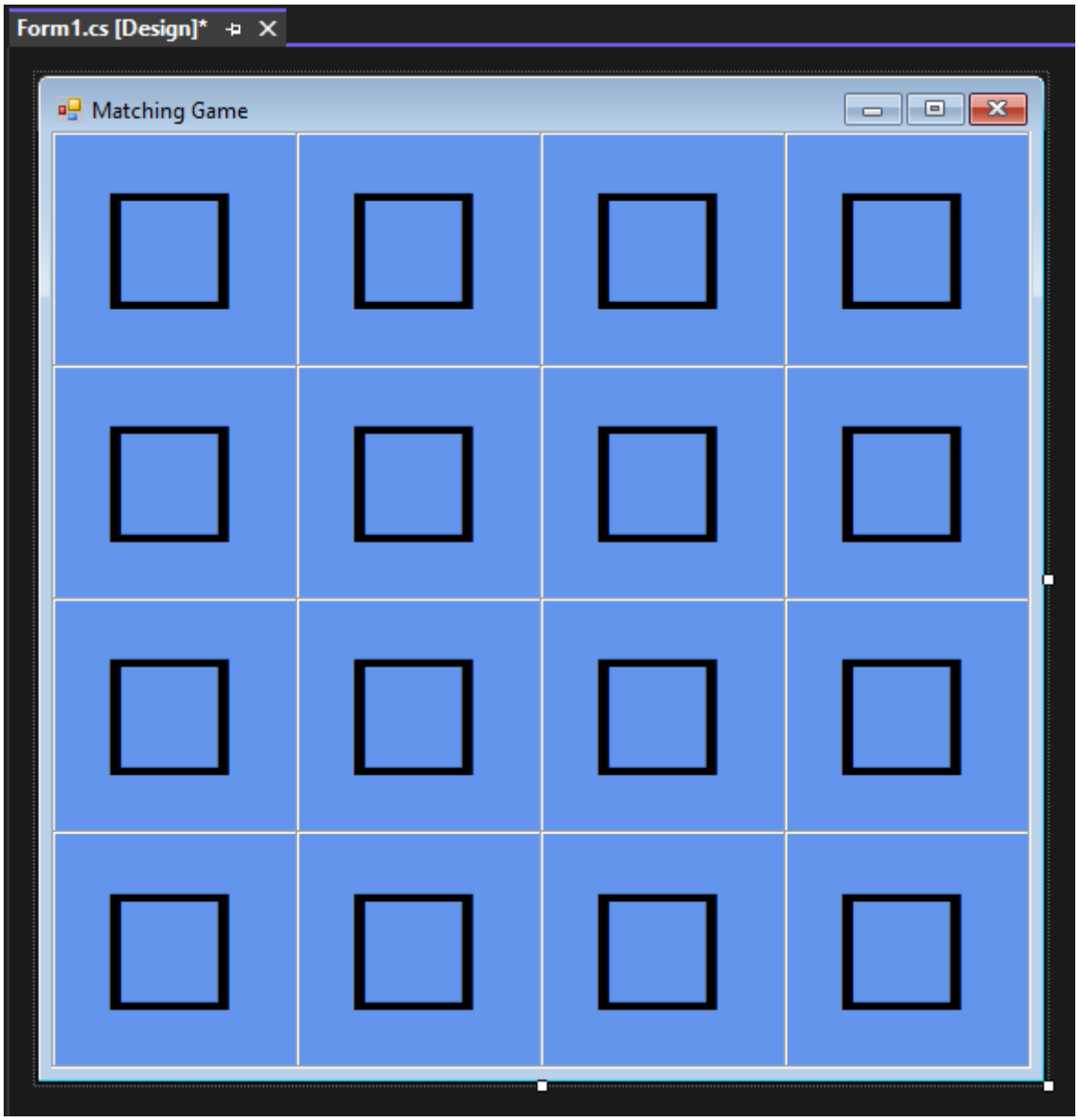
TableLayoutPanel 的左上角单元格现在包含一个在蓝色背景上居中的黑色框。

> **备注**
>
> Webdings 是 Windows 操作系统附带的图标字体。 在匹配游戏中，玩家匹配图标对。 此字体显示要匹配的图标。
>
> 在“Text”属性中尝试不同的字母，而不是 c。 感叹号是一个蜘蛛，大写 N 是一只眼，逗号是一个红辣椒。

3. 选择你的 Label 控件并将其复制到 TableLayoutPanel 中的下一单元格。 选择 Ctrl+C 键，或在菜单栏上依次选择“编辑”>“复制”。 然后，使用 Ctrl+V 或“编辑”>“粘贴”进行粘贴。

   第一个 Label 的副本将显示在 TableLayoutPanel 的第二个单元格中。 再次粘贴它，在第三个单元格中会出现另一 Label。 一直粘贴 Label 控件，直到填充完所有单元格。

此步将完成窗体的布局。



后续步骤

继续学习下一个教程，了解如何向每个标签分配随机图标，以及向标签添加事件处理程序。

向匹配游戏添加图标

教程：向匹配游戏 WinForms 应用中添加图标

项目 • 2023/03/23

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在这四个教程系列中，你将构建一个匹配游戏，玩家在其中匹配隐藏的图标对。

在匹配游戏中，玩家选择一个方块来查看图标，然后选择另一个方块。 如果图标互相匹配，则它们保持可见。 如果不匹配，游戏将隐藏这两个图标。 在本教程中，你将随机向标签分配图标。 将其设置为隐藏，然后在选中时显示。

在第二个教程中，你将了解如何：

- ✓ 添加 Random 对象和图标列表。
- ✓ 向每个标签分配一个随机图标。
- ✓ 添加事件处理程序，用于显示标签的图标。

必备条件

此教程基于上一教程创建匹配游戏应用程序。 如果尚未学习此教程，请先完成此教程。

添加 Random 对象和图标列表

在本部分中，你要为游戏创建一组匹配的符号。 每个符号将添加到窗体上 TableLayoutPanel 中的两个随机单元格。

使用 `new` 语句创建两个对象。 第一个是 Random 对象，随机选择 TableLayoutPanel 中的单元格。 第二个是 List<T> 对象。 它存储随机选择的符号。

1. 打开 Visual Studio。 MatchingGame 项目显示在“打开最近使用的文件”下。
2. 如果使用的是 C#，请选择“Form1.cs”，如果使用的是 Visual Basic，则选择“Form1.vb”。 然后选择“查看”>“代码”。 也可以选择 F7 键或双击“Form1”。 Visual Studio IDE 显示 Form1 的代码模块。
3. 在现有代码中，添加以下代码。

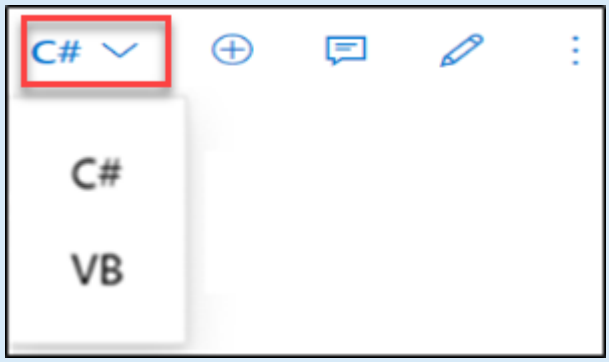
C#

```
C#
```

```
public partial class Form1 : Form
{
    // Use this Random object to choose random icons for the
squares
    Random random = new Random();

    // Each of these letters is an interesting icon
    // in the Webdings font,
    // and each icon appears twice in this list
    List<string> icons = new List<string>()
    {
        "!", "!", "N", "N", ",", ",", "k", "k",
        "b", "b", "v", "v", "w", "w", "z", "z"
    };
```

ⓘ 重要

使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



如果你使用 C#，请确保将代码放在左大括号后面，紧靠类声明 (`public partial class Form1 : Form`) 之后。 如果你使用 Visual Basic，请将代码放在紧靠类声明 (`Public Class Form1`) 之后。

可以使用列表对象跟踪不同类型的项目。 列表可以包含数字、true/false 值、文本或其他对象。 在匹配游戏中，列表对象具有 16 个字符串，每个字符串对应 TableLayoutPanel 面板中的每个单元格。 每个字符串都是对应于标签中的图标的单个字母。 这些字符在 Webdings 字体中显示为公共汽车、自行车和其他符号。

ⓘ 备注

列表可以根据需要收缩或增长，这在此程序中非常重要。

若要详细了解列表，请参阅 List<T>。 若要查看 C# 中的示例，请参阅基本列表示例。 若要查看 Visual Basic 中的示例，请参阅使用简单集合。

向每个标签分配一个随机图标

每次运行该程序时，它都会使用 `AssignIconsToSquares()` 方法将图标随机分配给 Label 控件。 此代码使用关键字 `foreach`（C# 中）或 `For Each`（Visual Basic 中）。

1. 添加 `AssignIconsToSquares()` 方法。

C#

C#

```
/// <summary>
/// Assign each icon from the list of icons to a random square
/// </summary>
private void AssignIconsToSquares()
{
    // The TableLayoutPanel has 16 labels,
    // and the icon list has 16 icons,
    // so an icon is pulled at random from the list
    // and added to each label
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;
        if (iconLabel != null)
        {
            int randomNumber = random.Next(icons.Count);
            iconLabel.Text = icons[randomNumber];
            // iconLabel.ForeColor = iconLabel.BackColor;
            icons.RemoveAt(randomNumber);
        }
    }
}
```

可以在上一节中添加的代码之下输入此代码。

> **备注**
>
> 其中有一行被故意注释掉了。 稍后在此过程中将其添加。

`AssignIconsToSquares()` 方法循环访问 TableLayoutPanel 中的每个标签控件。 它对每个控件运行相同的语句。 语句从列表中提取随机图标。

- 第一行将 control 变量转换为名为“iconLabel” 的标签。
- 第二行是 `if` 语句，用于检查以确保转换起作用。 如果转换起作用，则 `if` 语句中的语句将运行。
- `if` 语句中的第一行将创建一个名为“randomNumber”的变量，该变量包含一个与图标列表中的项对应的随机数。 它使用 Random 对象的 Next() 方法。 `Next` 方法将返回此随机数。 此行也使用“图标” 列表的 Count 属性来确定随机数的选择范围。
- 下一行会将图标列表项之一分配给标签的 Text 属性。
- 下一行将隐藏这些图标。 该行已在此处注释掉，因此你可以在继续操作之前验证代码的其余部分。
- `if` 语句中最后一行将从列表中删除已添加到窗体中的图标。

1. 将调用添加到 Form1 构造函数的 `AssignIconsToSquares()` 方法。 此方法用图标填充游戏板。 构造函数在创建对象时进行调用。

   C#

   ```
   public Form1()
   {
       InitializeComponent();

       AssignIconsToSquares();
   }
   ```

   对于 Visual Basic，将 `AssignIconsToSquares()` 方法调用添加到 `Form1_Load` 方法。

   VB

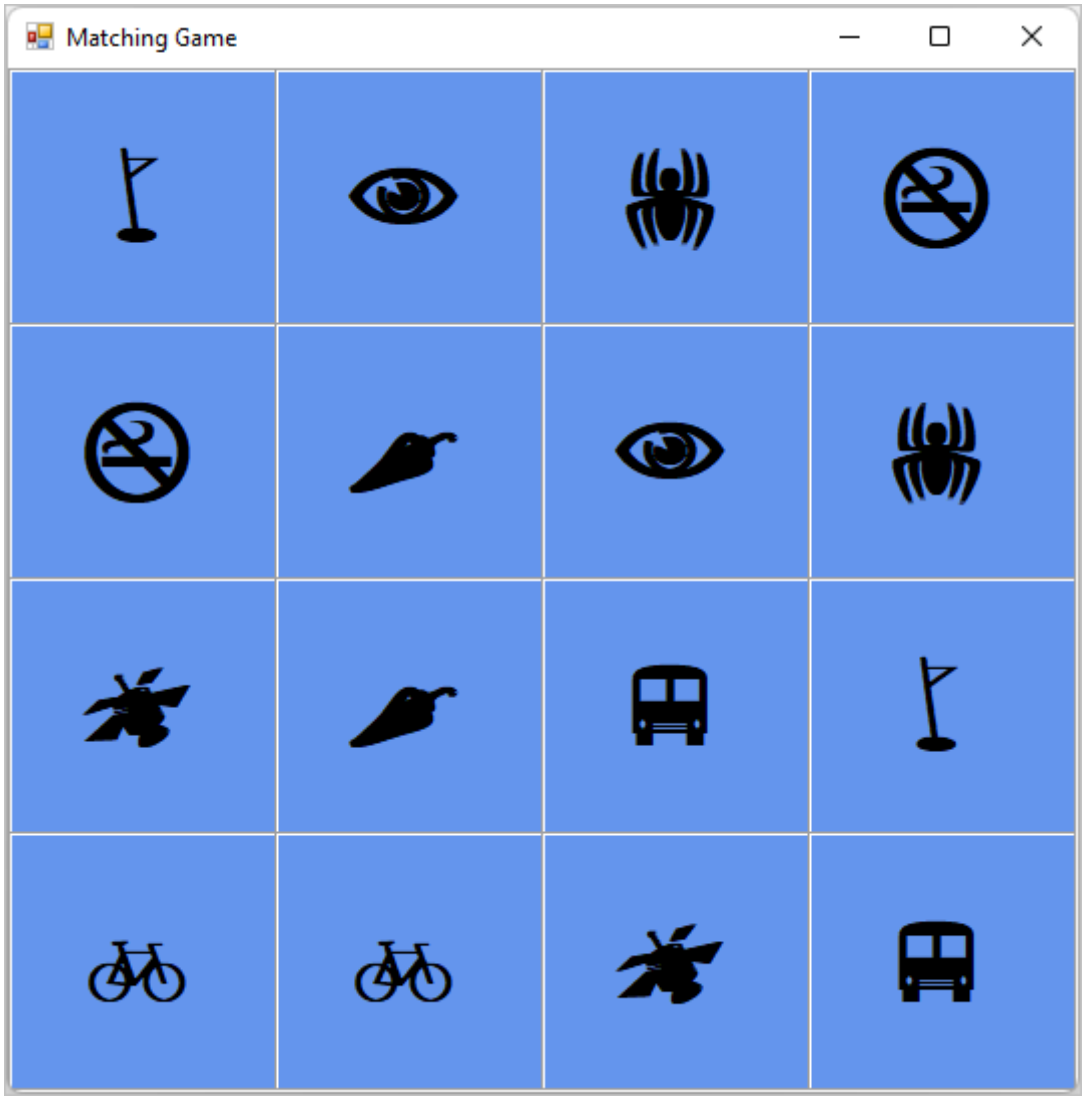
   ```
   Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
   MyBase.Load
       AssignIconsToSquares()
   End Sub
   ```

   有关详细信息，请参阅构造函数（C# 编程指南）或使用构造函数和析构函数。

2. 保存并运行程序。 它应该显示一个窗体，其中每个标签都分配了随机图标。

   > **提示**
   >
   > 如果窗体上的 Webdings 图标不能正确显示，请将窗体上标签的 UseCompatibleTextRendering 属性设置为 True。

3. 关闭程序，然后重新运行。 向每个标签分配不同的图标。



你没有隐藏这些图标，所以现在可以看到。 若要向玩家隐藏图标，可以将每个标签的“ForeColor” 属性设置为与“BackColor” 属性相同的颜色。

4. 停止程序。 删除循环内注释的代码行的注释标记。

C#

```
C#

iconLabel.ForeColor = iconLabel.BackColor;
```

如果再次运行该程序，图标似乎已消失。 只显示蓝色背景。 图标是随机分配的，仍然存在。

向标签添加事件处理程序

在此匹配游戏中，玩家先显示一个隐藏图标，然后再显示第二个图标。 如果图标互相匹配，则它们保持可见。 否则，两个图标都会再次隐藏。

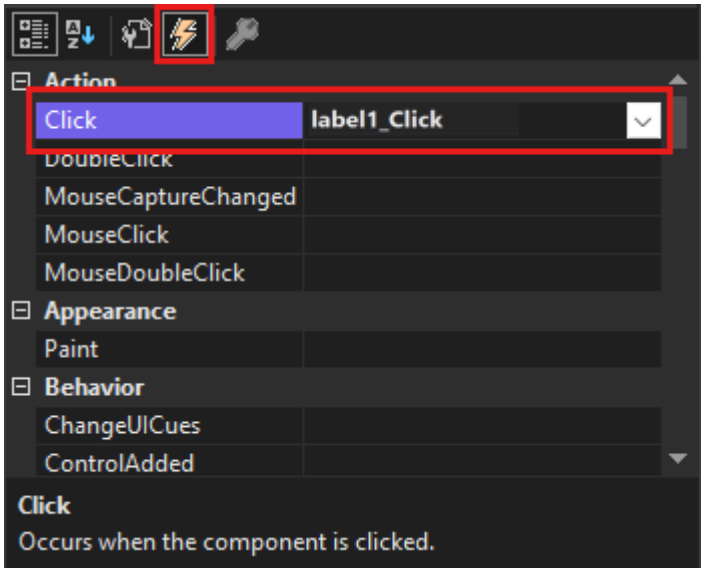
若要使游戏以这种方式工作，请添加一个 Click 事件处理程序，它将更改所选标签的颜色以与背景匹配。

1. 在“Windows 窗体设计器”中打开窗体。 选择“Form1.cs”或“Form1.vb”，然后选择“查看”>“设计器”。

2. 选择第一个标签控件以将其选中，然后双击它，将名为 label1 _Click() 的事件处理程序 `Click` 添加到代码中。

3. 然后，按住 Ctrl 键的同时选择其他每个标签。 确保选中每个标签。

4. 在“属性”窗口中，选择“事件”按钮，这是一个闪电形符号。 对于“Click”事件，请在框中选择“label1_Click” 。



5. 选择 **Enter** 键。 IDE 将名为 label1 _Click() 的 `Click` 事件处理程序添加到代码中。 由于你选择了所有标签，因此处理程序会与每个标签挂钩。

6. 填写其余代码。

C#

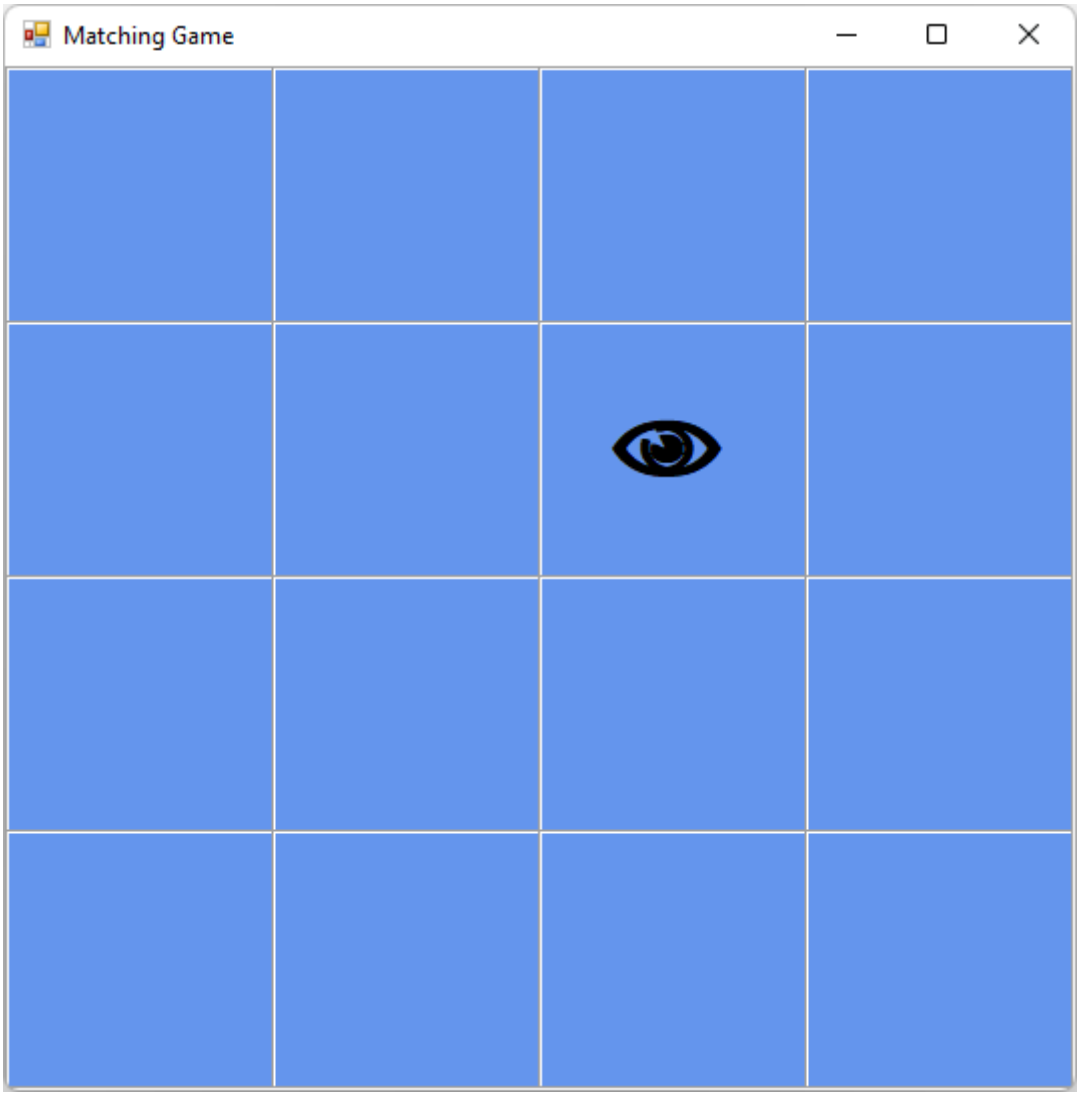
C#

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label1_Click(object sender, EventArgs e)
{
```

```
    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;

        clickedLabel.ForeColor = Color.Black;
    }
}
```

> **ⓘ 备注**
>
> 如果你复制和粘贴 `label1_Click()` 代码块，而不是手动输入代码，请确保替换现有的 `label1_Click()` 代码。 否则，你将得到重复的代码块。

选择“调试”>“开始调试”以运行程序。 您应该看到一个背景为蓝色的空窗体。 选择窗体中的任意单元格。 其中一个图标应可见。 继续在窗体中选择不同位置。 当选择图标时，它们应显示。



后续步骤

请继续学习下一教程，了解如何使用计时器更改标签。

在匹配游戏中使用计时器

教程：向匹配游戏 WinForms 应用中添加引用变量和计时器控件

项目 • 2024/01/13

在这四个教程系列中，你将构建一个匹配游戏，玩家在其中匹配隐藏的图标对。

匹配游戏程序需要跟踪玩家选择了哪些标签控件。 在玩家选择第一个标签后，该程序应显示图标。 在选择第二个标签后，该程序应短暂显示两个图标。 然后隐藏这两个图标。

程序通过引用变量跟踪第一次和第二次分别选择的标签控件。 计时器隐藏图标并控制显示图标的时间长度

- ✓ 添加标签引用。
- ✓ 添加计时器。

必备条件

本教程基于前面的教程，即创建匹配游戏应用程序和向匹配游戏中添加图标。 请先完成这些教程的学习。

添加标签引用

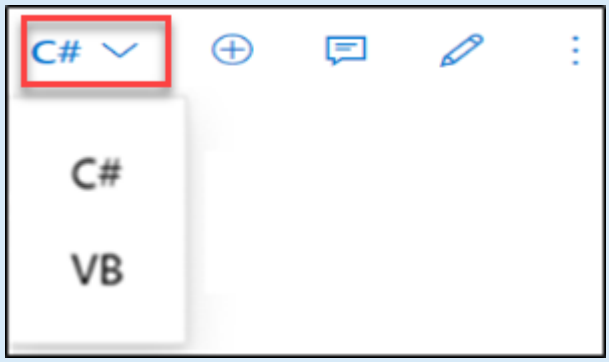
本部分将向代码添加两个引用变量。 它们跟踪或引用标签对象。

1. 通过使用下面的代码向窗体中添加标签引用。

C#

```
public partial class Form1 : Form
{
    // firstClicked points to the first Label control
    // that the player clicks, but it will be null
    // if the player hasn't clicked a label yet
    Label firstClicked = null;

    // secondClicked points to the second Label control
    // that the player clicks
    Label secondClicked = null;
```

> **ⓘ 重要**
>
> 使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



这些语句不会导致窗体中显示标签控件，因为没有 `new` 关键字。 当程序启动时，`firstClicked` 和 `secondClicked` 都设置为 `null`（对于 C#）或 `Nothing`（对于 Visual Basic）。

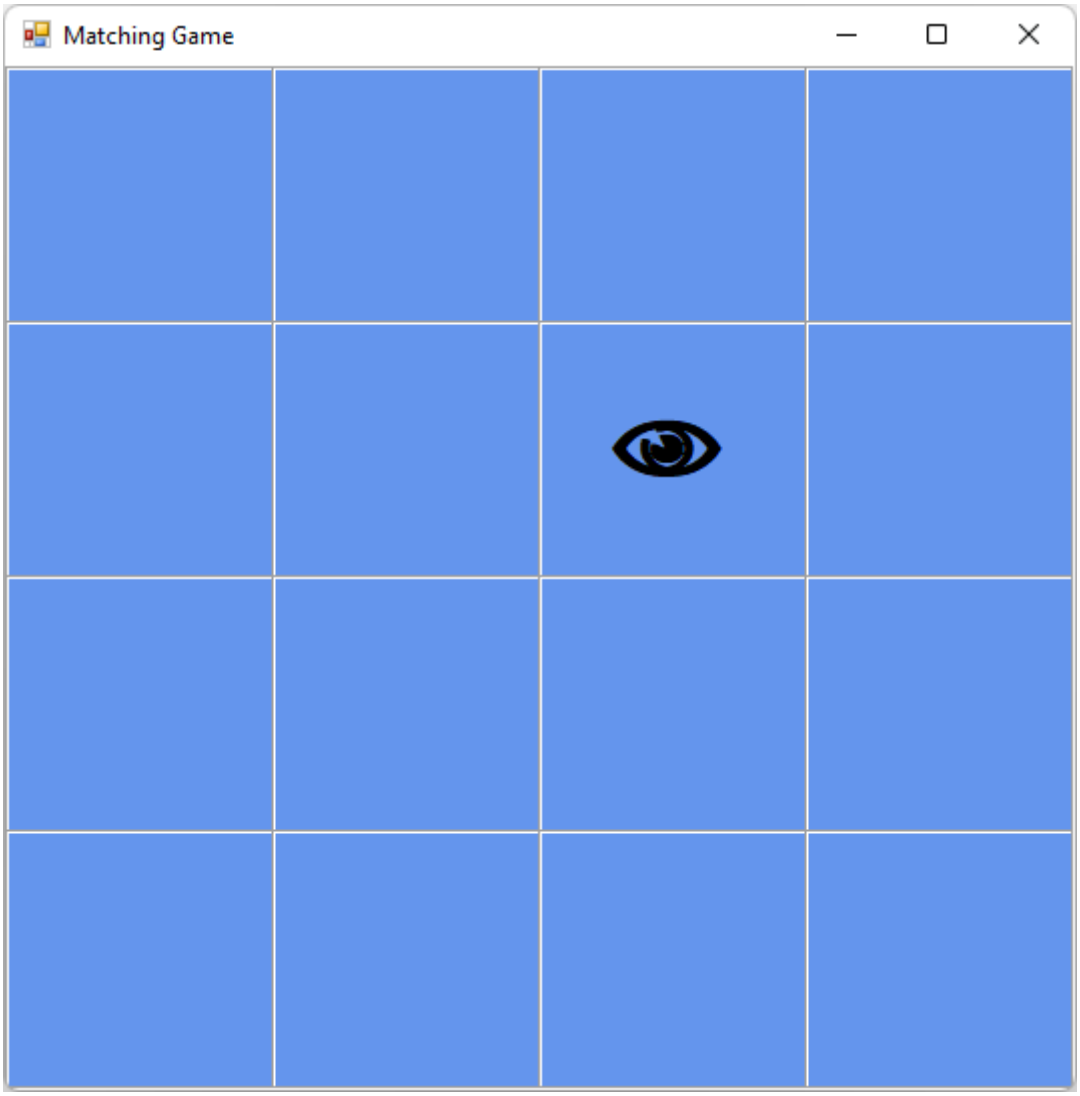
2. 修改 Click 事件处理程序，以使用新的 `firstClicked` 引用变量。 移除 `label1_Click()` 事件处理程序方法中的最后一个语句（`clickedLabel.ForeColor = Color.Black;`），并将它替换为下面的 `if` 语句。

C#

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label1_Click(object sender, EventArgs e)
{
    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;

        // If firstClicked is null, this is the first icon
        // in the pair that the player clicked,
        // so set firstClicked to the label that the player
        // clicked, change its color to black, and return
        if (firstClicked == null)
        {
```

```
            firstClicked = clickedLabel;
            firstClicked.ForeColor = Color.Black;

            return;
        }
    }
}
```

3. 保存并运行程序。 选择其中一个标签控件，它的图标将显示。 选择下一个标签控件，发现没有任何反应。



只有选择的第一个图标会出现。 其他图标是不可见的。

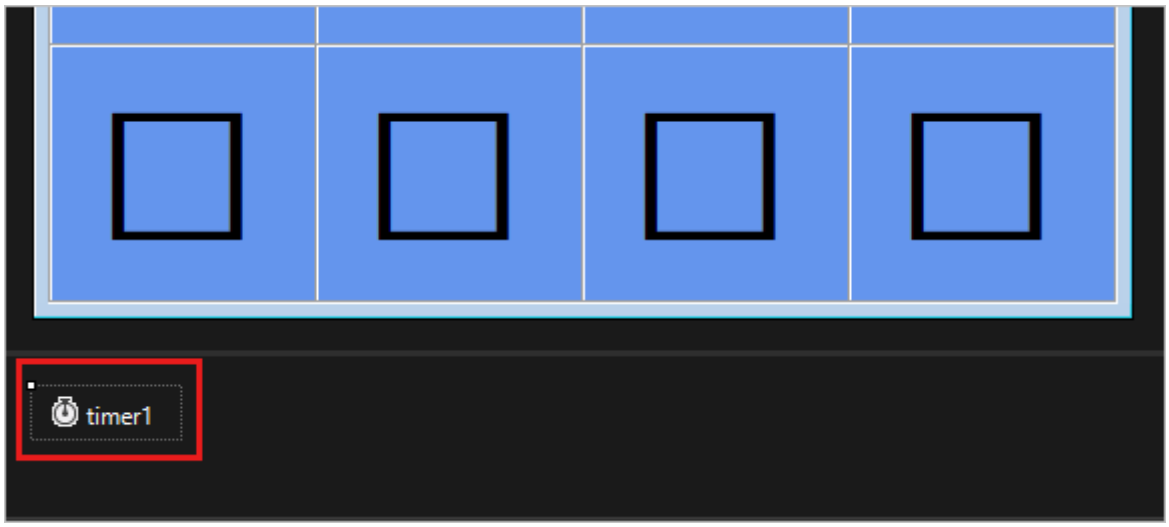
该程序已跟踪玩家选择的第一个标签。 引用 `firstClicked` 不是 C# 中的 `null` 或 Visual Basic 中的 `Nothing`。 当 `if` 语句发现 `firstClicked` 不等于 `null` 或 `Nothing` 时，会运行这些语句。

添加计时器

匹配游戏应用使用 Timer 控件。 计时器等待后，触发一个称为“Tick”的事件。 计时器可以启动操作或定期重复操作。

在程序中，计时器允许玩家选择两个图标。 如果图标不匹配，则在短暂时间后再次隐藏这两个图标。

1. 选择“工具箱”选项卡，在“组件”类别中，双击“计时器”组件或将其拖到窗体中。 该计时器图标名为“timer1”，显示在窗体下的空间中。



2. 选择“Timer1”图标以选中该计时器。 在“属性”窗口中，选择“属性”按钮以查看属性。

3. 将“间隔”属性设置为“750”，即 750 毫秒。

   “间隔”属性将通知计时器两个时钟周期之间的等待时长，或何时触发 Tick 事件。 在玩家选择第二个标签后，程序调用 Start() 方法启动计时器。

4. 选择计时器控件图标，然后按 Enter，或双击计时器。 IDE 添加空的 Tick 事件处理程序。 将代码替换为以下代码。

C#

```
/// <summary>
/// This timer is started when the player clicks
/// two icons that don't match,
/// so it counts three quarters of a second
/// and then turns itself off and hides both icons
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void timer1_Tick(object sender, EventArgs e)
{
    // Stop the timer
    timer1.Stop();

    // Hide both icons
```

```
    firstClicked.ForeColor = firstClicked.BackColor;
    secondClicked.ForeColor = secondClicked.BackColor;

    // Reset firstClicked and secondClicked
    // so the next time a label is
    // clicked, the program knows it's the first click
    firstClicked = null;
    secondClicked = null;
}
```

Tick 事件处理程序将执行三项操作：

- 通过调用 Stop() 方法确保计时器没有运行。
- 它使用两个引用变量 `firstClicked` 和 `secondClicked`，使得玩家选择的两个标签的图标再次不可见。
- 它将 `firstClicked` 和 `secondClicked` 引用变量重置为 `null`（C# 中）或 `Nothing`（Visual Basic 中）。

5. 转至代码编辑器，将代码添加到 `label1_Click()` 事件处理程序方法的顶部和底部。此代码将检查计时器是否已启用，设置 `secondClicked` 引用变量，并启动计时器。`label1_Click()` 事件处理程序方法现在如下所示：

C#

```
/// <summary>
/// Every label's Click event is handled by this event handler
/// </summary>
/// <param name="sender">The label that was clicked</param>
/// <param name="e"></param>
private void label1_Click(object sender, EventArgs e)
{
    // The timer is only on after two non-matching
    // icons have been shown to the player,
    // so ignore any clicks if the timer is running
    if (timer1.Enabled == true)
        return;

    Label clickedLabel = sender as Label;

    if (clickedLabel != null)
    {
        // If the clicked label is black, the player clicked
        // an icon that's already been revealed --
        // ignore the click
        if (clickedLabel.ForeColor == Color.Black)
            return;
```

```
        // If firstClicked is null, this is the first icon
        // in the pair that the player clicked,
        // so set firstClicked to the label that the player
        // clicked, change its color to black, and return
        if (firstClicked == null)
        {
            firstClicked = clickedLabel;
            firstClicked.ForeColor = Color.Black;
            return;
        }

        // If the player gets this far, the timer isn't
        // running and firstClicked isn't null,
        // so this must be the second icon the player clicked
        // Set its color to black
        secondClicked = clickedLabel;
        secondClicked.ForeColor = Color.Black;

        // If the player gets this far, the player
        // clicked two different icons, so start the
        // timer (which will wait three quarters of
        // a second, and then hide the icons)
        timer1.Start();
    }
}
```

- 该方法顶部的代码通过检查 **Enabled** 属性的值来检查计时器是否已启动。 如果玩家选择第一个和第二个标签控件，且计时器启动，则选择第三个控件将不会执行任何操作。
- 该方法底部的代码将 `secondClicked` 引用变量设置为跟踪第二个标签控件。 然后将该标签的图标颜色设置为黑色以使其可见。 然后，它在单触发模式下启动计时器，以便在等待 750 毫秒后触发单个 Tick。 计时器的 Tick 事件处理程序会隐藏这两个图标，并重置 `firstClicked` 和 `secondClicked` 引用变量。 窗体准备就绪供玩家选择另一对图标。

> **ⓘ 备注**
>
> 如果你复制和粘贴 `label1_Click()` 代码块，而不是手动输入代码，请确保替换现有的 `label1_Click()` 代码。 否则，你将得到重复的代码块。

6. 保存并运行程序。 选择一个正方形，图标将变得可见。 选择另一个正方形。 图标会短暂显示，然后两个图标都消失。

程序现在跟踪你选择的第一个和第二个图标。 它使用计时器在使图标消失之前暂停。

后续步骤

请继续阅读下一篇教程，了解如何完成匹配游戏。

为匹配游戏显示祝贺消息

教程：在匹配游戏 WinForms 应用中显示消息

项目 • 2024/01/13

在这四个教程系列中，你将构建一个匹配游戏，玩家在其中匹配隐藏的图标对。

在本教程中，你将修改匹配游戏以保持匹配的对可见并在玩家获胜时显示祝贺消息。

在本教程中，你将了解如何执行以下操作：

- ✔ 保持对可见。
- ✔ 验证玩家是否获胜。
- ✔ 试用其他功能。

必备条件

本教程是在以前这些教程的基础上编写的：

1. 创建匹配游戏应用程序
2. 向匹配游戏添加图标
3. 在匹配游戏中添加计时器

保持对可见

当玩家匹配对时，游戏应当进行重置，这样游戏就不再跟踪使用 `firstClicked` 和 `secondClicked` 引用变量的任何标签。 不应重置匹配的两个标签的颜色。 这些标签将继续显示。

1. 将下面的 `if` 语句添加到 `label_Click()` 事件处理程序方法中。 将它放在紧靠启动计时器的语句上方代码的结尾处。

C#

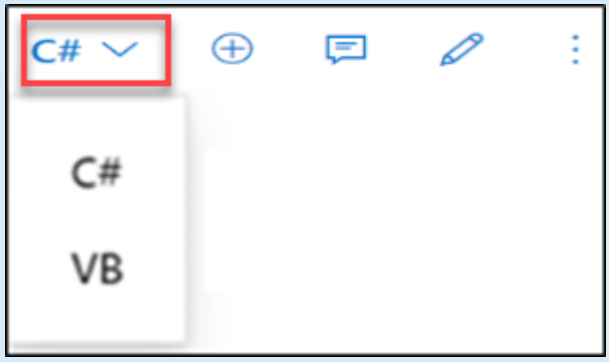
C#

```
        // If the player gets this far, the timer isn't
        // running and firstClicked isn't null,
        // so this must be the second icon the player clicked
        // Set its color to black
        secondClicked = clickedLabel;
        secondClicked.ForeColor = Color.Black;
```

```
        // If the player clicked two matching icons, keep them
        // black and reset firstClicked and secondClicked
        // so the player can click another icon
        if (firstClicked.Text == secondClicked.Text)
        {
            firstClicked = null;
            secondClicked = null;
            return;
        }

        // If the player gets this far, the player
        // clicked two different icons, so start the
        // timer (which will wait three quarters of
        // a second, and then hide the icons)
        timer1.Start();
    }
}
```

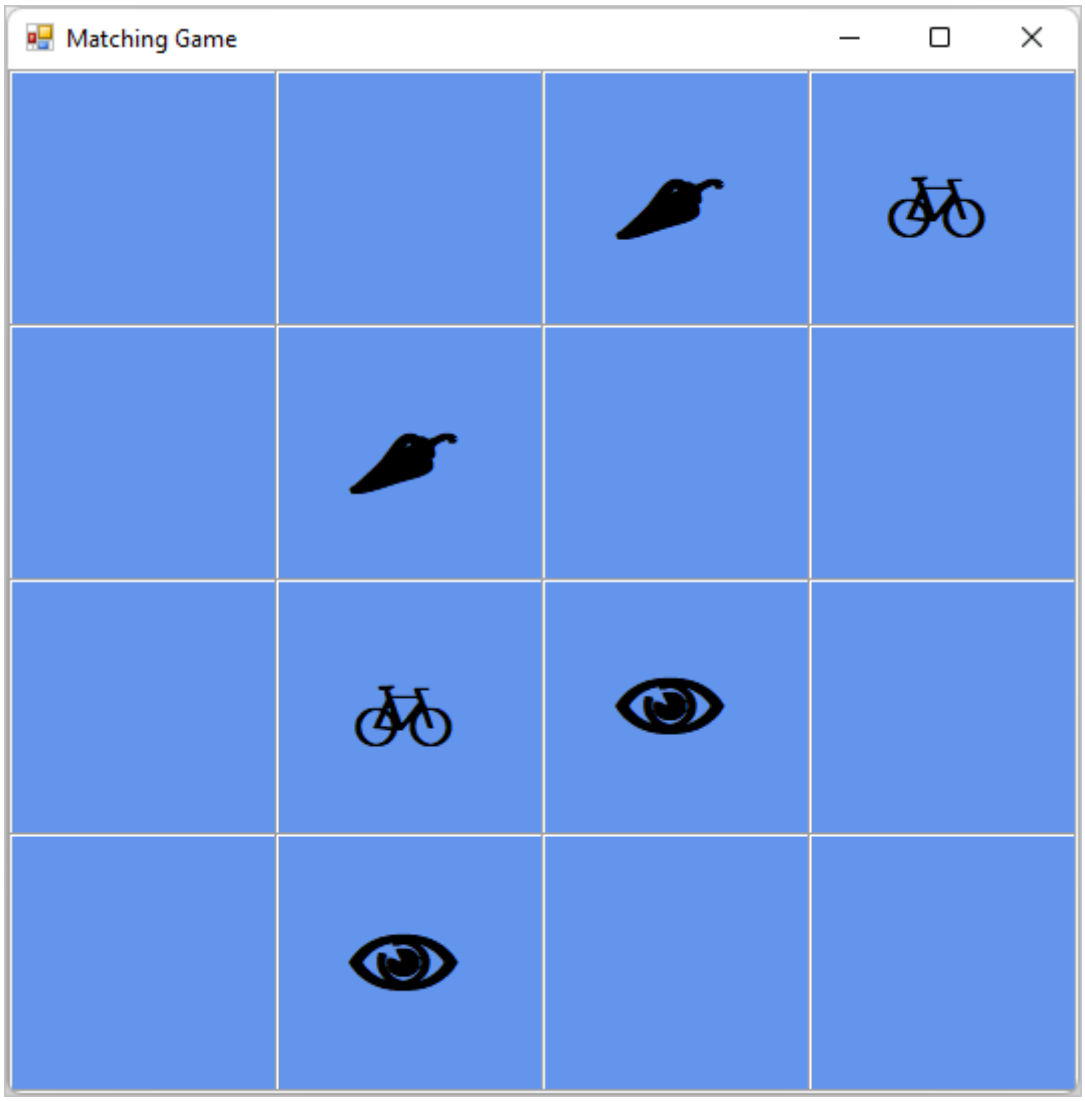
ⓘ 重要

使用此页右上角的编程语言控件查看 C# 代码片段或 Visual Basic 代码片段。



`if` 语句会检查玩家选择的第一个标签中的图标是否与第二个标签中的图标相同。 如果图标相同，则程序运行其三个语句。 前两个语句重置 `firstClicked` 和 `secondClicked` 引用变量。 它们不再跟踪任何标签。 第三个语句是 `return` 语句，它跳过方法中的其余语句，不运行它们。

2. 运行程序，然后开始选择窗体上的正方形。



如果选择的是不匹配的对，则将触发计时器的 Tick 事件。 两个图标都会消失。

如果选择的是匹配的对，则将运行新的 `if` 语句。 return 语句会使方法跳过启动计时器的代码。 因此图标保持可见。

验证玩家是否获胜

你已创建了一个有趣的游戏。 玩家获胜后，游戏应结束。 本部分添加验证玩家是否获胜的方法。

1. 在你的代码底部，`timer1_Tick()` 事件处理程序下方添加一个 `CheckForWinner()` 方法。

C#

C#

```
/// <summary>
/// Check every icon to see if it is matched, by
/// comparing its foreground color to its background color.
```

```
/// If all of the icons are matched, the player wins
/// </summary>
private void CheckForWinner()
{
    // Go through all of the labels in the TableLayoutPanel,
    // checking each one to see if its icon is matched
    foreach (Control control in tableLayoutPanel1.Controls)
    {
        Label iconLabel = control as Label;

        if (iconLabel != null)
        {
            if (iconLabel.ForeColor == iconLabel.BackColor)
                return;
        }
    }

    // If the loop didn’t return, it didn't find
    // any unmatched icons
    // That means the user won. Show a message and close the form
    MessageBox.Show("You matched all the icons!", "Congratulations");
    Close();
}
```

该方法使用 C# 中的另一个 `foreach` 循环或Visual Basic 中的 `For Each` 循环以执行 TableLayoutPanel 中的每个标签。 它会检查每个标签的图标颜色，以验证它是否与背景匹配。 如果颜色匹配，图标将保持不可见，玩家还没有匹配所有剩余的图标。

在这种情况下，程序使用 `return` 语句跳过其余方法。 如果循环遍历所有标签而不执行 `return` 语句，则意味着窗体上的所有图标均已匹配。 该程序将显示一个恭喜玩家获胜的 MessageBox，然后调用 `Close()` 方法来结束游戏。

2. 让标签的 Click 事件处理程序调用新的 `CheckForWinner()` 方法。

C#

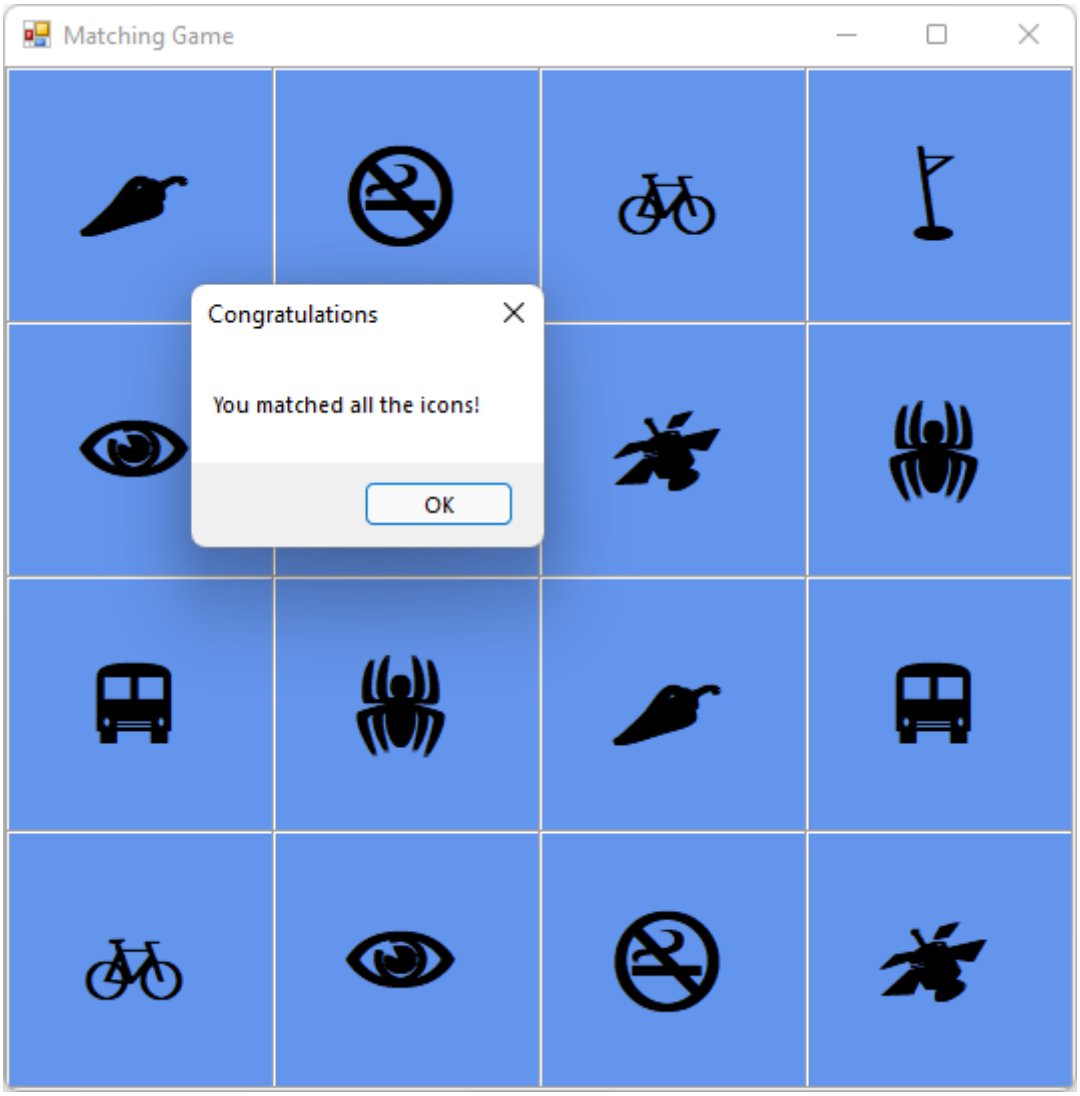
C#

```
// If the player gets this far, the timer isn't
// running and firstClicked isn't null,
// so this must be the second icon the player clicked
// Set its color to black
secondClicked = clickedLabel;
secondClicked.ForeColor = Color.Black;

// Check to see if the player won
CheckForWinner();

// If the player clicked two matching icons, keep them
```

```
// black and reset firstClicked and secondClicked
// so the player can click another icon
if (firstClicked.Text == secondClicked.Text)
{
    firstClicked = null;
    secondClicked = null;
    return;
}
```

请确保程序在显示玩家选择的第二个图标后立即检查是否有赢家。 查找设置第二个选中图标颜色的行，然后在该行后直接调用 `CheckForWinner()` 方法。

3. 保存并运行程序。 玩游戏并匹配所有图标。 当你获胜时，该程序将显示一个祝贺性的消息。



选择“确定”后，匹配游戏将关闭。

尝试其他功能

匹配游戏已完成。 你可以添加更多功能，使此游戏更具挑战性且更有趣。 提供以下选择。

- 将图标和颜色替换为您选择的图标和颜色。

  尝试查看标签的 Forecolor 属性。

- 添加一个游戏计时器，记录玩家赢得游戏所用的时间。

  可以在窗体中添加一个标签来显示经过的时间。 将其置于 TableLayoutPanel 上方。 在窗体中添加另一个计时器来记录时间。 使用代码在玩家开始游戏时启动计时器，并在最后两个图标匹配成功后停止计时器。

- 当玩家找到匹配的图标时会添加一种声音，当玩家发现两个不匹配的图标时会添加另一种声音，当程序再次隐藏图标时会添加第三种声音。

  若要播放声音，可以使用 System.Media 命名空间。 有关详细信息，请参阅在 Windows 窗体应用中播放声音 (C#) 或如何在 Visual Basic 中播放音频。

- 通过将图板变大，增加游戏的难度。

  不仅需要向 TableLayoutPanel 中添加行和列， 还需要考虑创建的图标数目。

- 如果玩家反应太慢，则隐藏第一个图标，以使游戏更具挑战性。

后续步骤

祝贺！ 你已完成该系列教程。 你已在 Visual Studio IDE 中完成以下编程和设计任务：

- 已在列表中存储对象，例如图标
- 已使用 C# 或 Visual Basic 中的循环来循环访问列表
- 已使用引用变量跟踪状态
- 已生成事件处理程序，以响应多个对象的事件
- 已添加进行倒计时并触发事件的计时器

进入本文深入了解 Windows 窗体设计器。

教程：Windows 窗体设计器入门

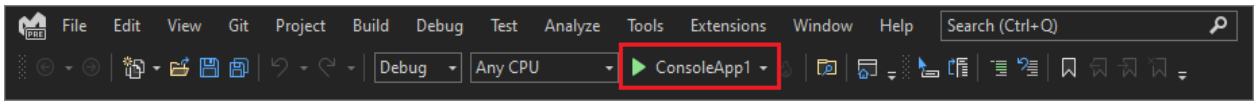
如何：在 Visual Studio 中运行 C# 程序

项目 • 2023/03/16

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

如何运行程序取决于你从什么开始执行、程序的类型，以及你是否想在调试器下运行。 最简单的是在 Visual Studio 中生成并运行一个打开的项目，在这种情况下，可执行以下操作：

- 按 F5，从 Visual Studio 菜单中选择“调试”>“开始执行(调试)”，或在 Visual Studio 工具栏上选择绿色的“开始”箭头和项目名称 。
- 你也可以按 Ctrl+F5，或从 Visual Studio 菜单中选择“调试”>“开始执行(不调试)”，直接运行而不调试 。

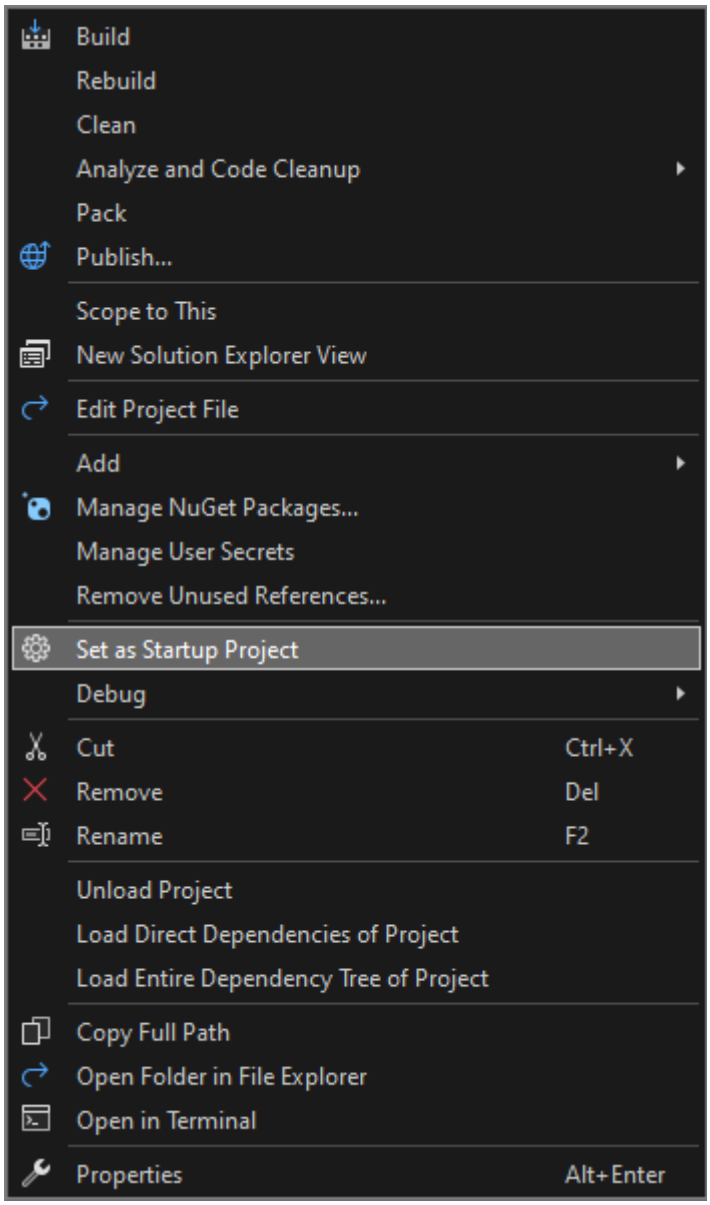


从项目开始

你可以运行 C# 项目或 .csproj 文件（如果是可运行的程序）。 如果项目包含带有 `Main` 方法的 C# 文件，并且其输出为可执行文件（.exe 文件），则很可能会在项目成功生成后运行该文件。 较新版本的 C# 不需要 `Main` 方法；相反，程序执行从顶级语句开始。 请参阅不含 Main 方法的程序。

1. 如果你的程序代码已在 Visual Studio 项目中，请打开该项目。 你可以执行以下操作来打开项目：双击或点击 Windows 文件资源管理器中的 .csproj 文件；或者在 Visual Studio 中选择“打开项目”，浏览以找到 .csproj 文件，然后选择该文件。

2. 在 Visual Studio 中加载项目后，如果 Visual Studio 解决方案包含多个项目，请确保将带有 `Main` 方法的项目设置为启动项目。 要设置启动项目，请右键单击“解决方案资源管理器”中的项目名称或节点，然后从上下文菜单中选择“设置为启动项目” 。



3. 要运行程序，请按 Ctrl+F5，从顶部菜单中选择“调试”>“开始执行(不调试)”，或选择绿色的“开始”按钮 。

   Visual Studio 会尝试生成并运行你的项目。 在 Visual Studio 屏幕的底部，生成输出显示在“输出”窗口中，生成错误显示在“错误列表”窗口中 。

   如果生成成功，应用会以适合项目类型的方式运行。 控制台应用在终端窗口中运行，Windows 桌面应用在新的桌面窗口中启动，Web 应用在由 IIS Express 承载的浏览器中运行。

从代码开始

如果是从代码清单、代码文件或少量文件开始，请首先确保该代码来自受信任的源，并且是可运行的程序。 具有 `Main` 方法的任何应用都可能是可运行的程序，但对于当前版本的 C#，不含 `Main` 方法但具有顶级语句的程序也可以运行。 你可以使用“控制台应用程序”模板创建一个项目，以便在 Visual Studio 中使用该应用。

单个文件的代码列表

1. 启动 Visual Studio，并打开一个空的 C# 控制台应用程序项目。
2. 将项目 .cs 文件中的所有代码替换为代码清单或文件的内容。
3. 将项目 .cs 文件重命名为与你的代码文件名一致。

磁盘上有多个代码清单或文件

1. 启动 Visual Studio，并创建适当类型的新项目。 如果不确定，请使用 C#“控制台应用程序”。
2. 在新项目中，将项目代码文件中的所有代码替换为第一个代码清单或文件的内容。
3. 重命名项目代码文件，使其与你的代码文件名一致。
4. 对于其余的每个代码文件：
   a. 右键单击“解决方案资源管理器”中的项目节点，然后选择“添加”>“现有项”，或选择该项目，然后按 Shift+Alt+A 。
   b. 浏览到并选择要导入到项目中的代码文件。

文件夹中有多个文件

如果文件夹中有多个文件，请先查找项目或解决方案文件。 Visual Studio 创建的程序具有项目和解决方案文件。 在 Windows 文件资源管理器中，查找扩展名为 .csproj 或 .sln 的文件 。 双击 .csproj 文件，在 Visual Studio 中打开它。 请参阅从 Visual Studio 解决方案或项目开始。

如果代码来自其他开发环境，则没有项目文件。 在 Visual Studio 中选择“打开”>“文件夹”，打开该文件夹 。 请参阅开发代码而无需创建项目或解决方案。

从 GitHub 或 Azure DevOps 存储库开始

如果要运行的代码位于 GitHub 或 Azure DevOps 存储库中，则可以使用 Visual Studio 直接从存储库打开项目。 请参阅打开存储库中的项目。

运行程序

要开始生成程序，请按 Visual Studio 工具栏上的绿色“开始”按钮，或者按 F5 或 Ctrl+F5 。 使用“开始”按钮或 F5 在调试器下运行程序 。

Visual Studio 会尝试在你的项目中生成并运行代码。 如果生成不成功，请参阅以下部分，了解有关如何成功生成项目的一些建议。

疑难解答

你的代码可能有错误。 或者代码可能是正确的，但它可能依赖于缺少的程序集或 NuGet 包，或者针对其他版本的 .NET。 在这些情况下，可以轻松地修复生成。

添加引用

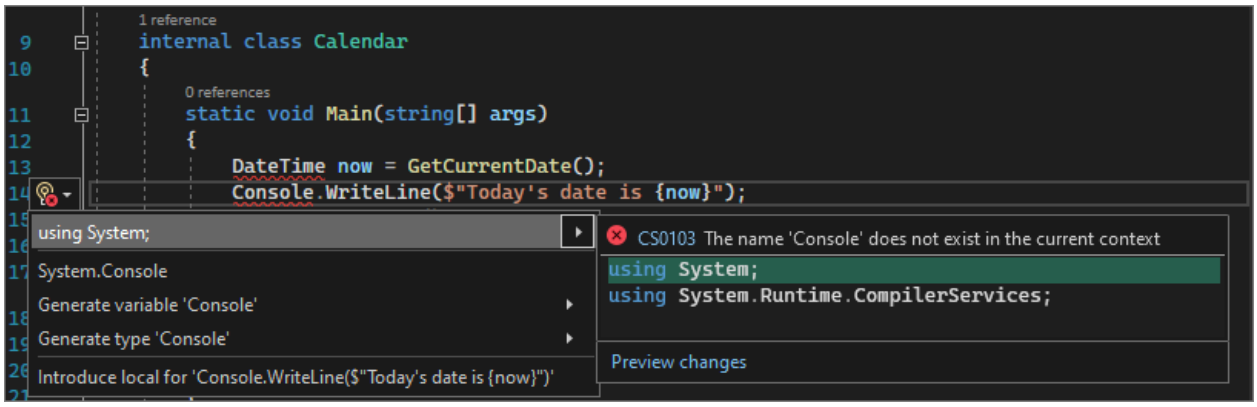
要正确生成，代码必须正确，并具有对库或其他依赖项的正确引用。 代码中的红色波浪下划线或错误列表中的条目表明存在错误，即使在编译和运行程序之前也是如此。 如果错误与未解析的名称相关，则可能需要添加引用和/或 `using` 指令。 如果代码引用了任何缺失的程序集或 NuGet 包，则需要将这些引用添加到项目。

Visual Studio 会尝试帮助你识别缺少的引用。 如果名称未解析，编辑器中会出现灯泡图标。 选择灯泡，可以看到有关如何修复此问题的一些建议。 解决方法可能是：

- 添加 using 指令。
- 添加对程序集的引用。
- 安装 NuGet 包。

添加 using 指令

下面是缺少的 `using` 指令的示例。 你可以将 `using System;` 添加到代码文件的开头，以解析无法解析的名称 `Console`：

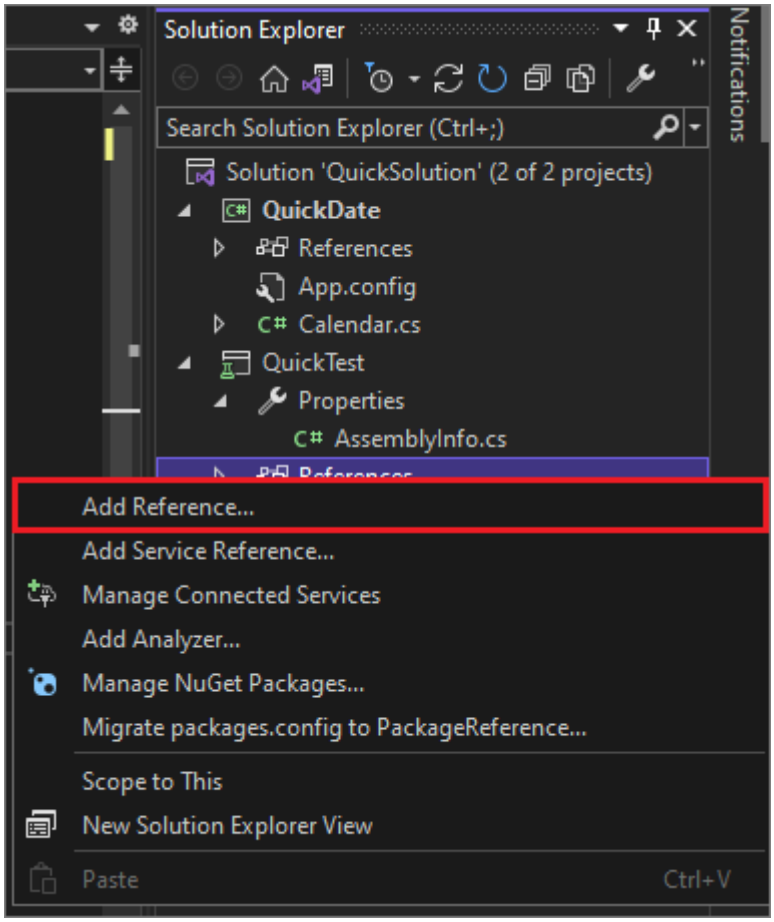


较新版本的 C# 支持对一些常用命名空间使用隐式 using 指令，因此，如果在创建项目时选择了该选项，则不需要它们。

添加程序集引用

.NET 引用可以是程序集或 NuGet 包。 在源代码中，发布者或作者通常会说明代码所需的程序集以及它所依赖的包。 要手动添加对项目的引用，请右键单击“解决方案资源管理

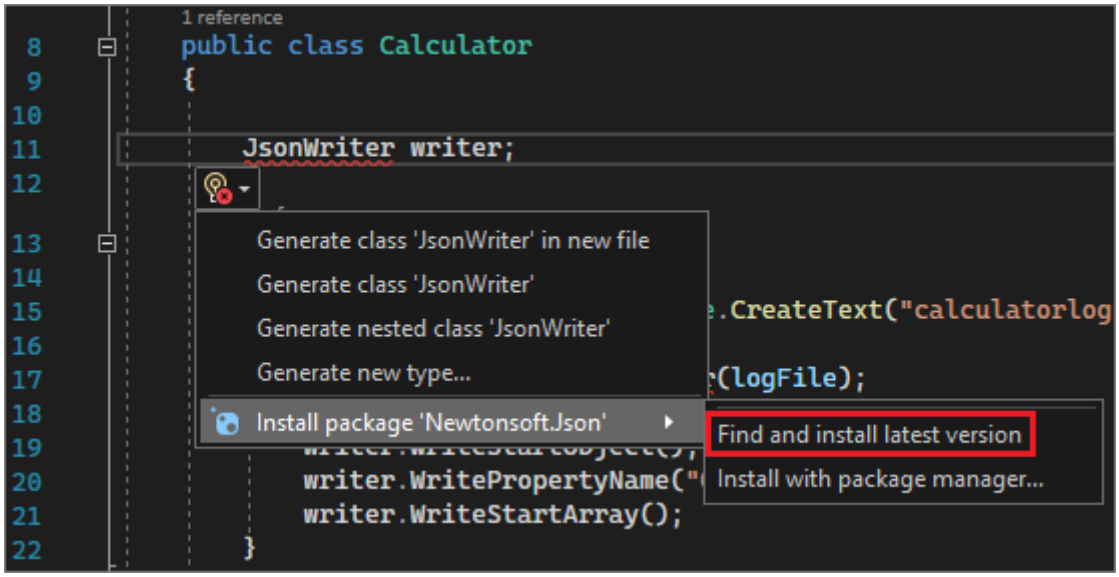
器”中的“引用”节点，然后选择“添加引用” 。 在“引用管理器”中，找到并添加所需的程序集。



你可以按照使用引用管理器添加或删除引用中的说明查找程序集并添加引用。

添加 NuGet 包

如果 Visual Studio 检测到缺少 NuGet 包，则会显示一个灯泡，并提供安装包的选项：



如果这不能解决问题，或者 Visual Studio 找不到包，请尝试联机搜索包。 请参阅在 Visual Studio 中安装和使用 NuGet 包。

使用正确版本的 .NET

由于不同版本的 .NET Framework 具有一定的后向兼容性，因此较新的框架可能会运行针对较旧的框架编写的代码，且不需要进行任何更改。 但有时需要针对特定的 .NET Framework 版本。 你可能需要安装特定版本的 .NET Framework 或 .NET Core。 请参阅修改 Visual Studio。

要更改目标 .NET Framework 版本，请参阅更改目标框架。 有关详细信息，请参阅 .NET Framework 目标错误疑难解答。

后续步骤

- 阅读欢迎使用 Visual Studio IDE，探索 Visual Studio 开发环境。
- 创建自己的第一个 C# 应用

教程：打开存储库中的项目

项目 • 2023/12/04

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在本教程中，你将使用 Visual Studio 首次连接到存储库，进行克隆，然后打开其中的一个项目。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

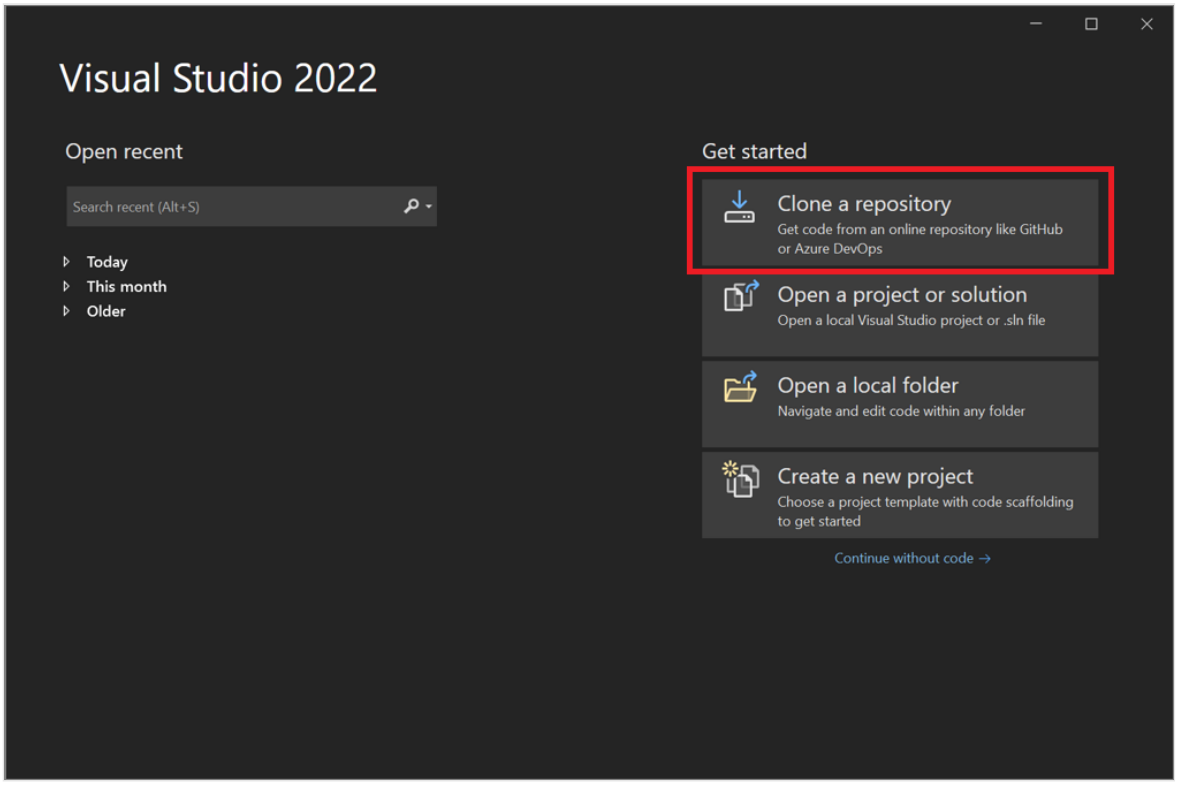
打开 GitHub 存储库中的项目

可以使用 Visual Studio 轻松打开存储库中的项目。 可以在启动 Visual Studio 时执行此操作，也可以直接从 Visual Studio IDE 中执行此操作。

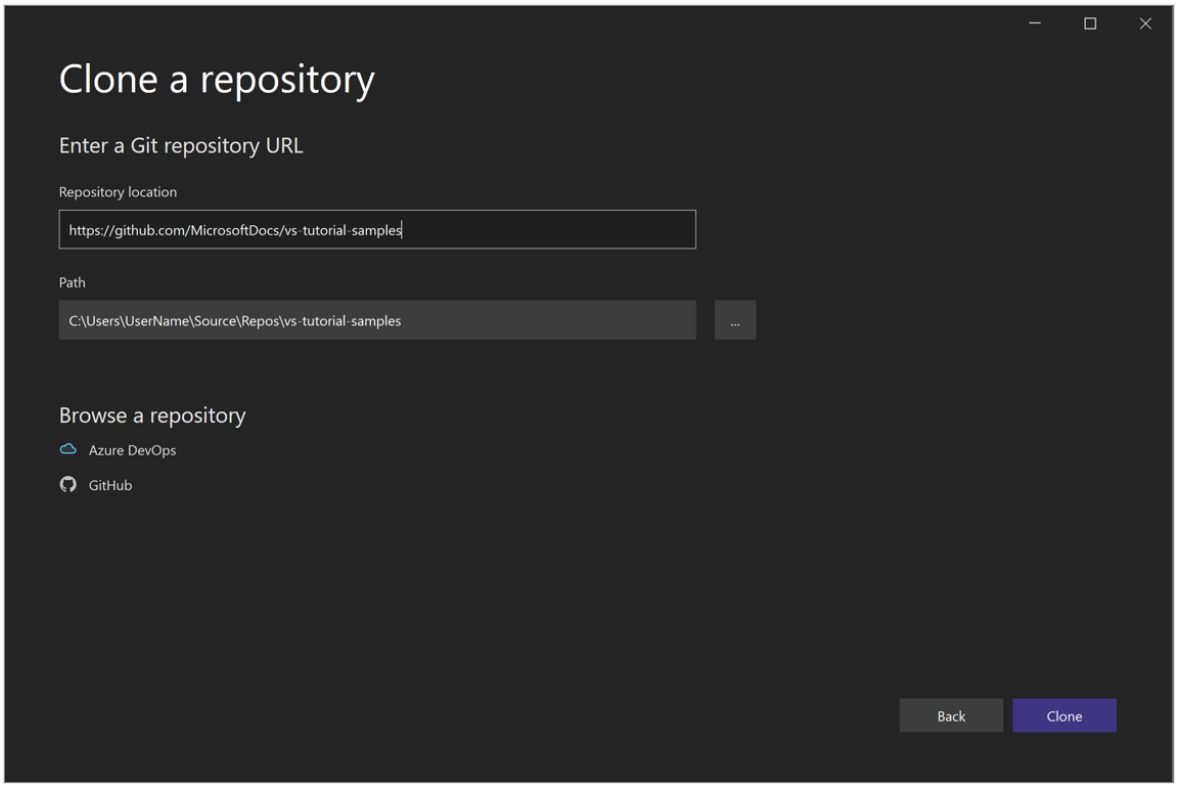
操作方法如下。

使用“开始”窗口

1. 打开 Visual Studio。
2. 在起始窗口中，选择“克隆存储库”。



3. 输入或键入存储库位置，然后选择“克隆”按钮。

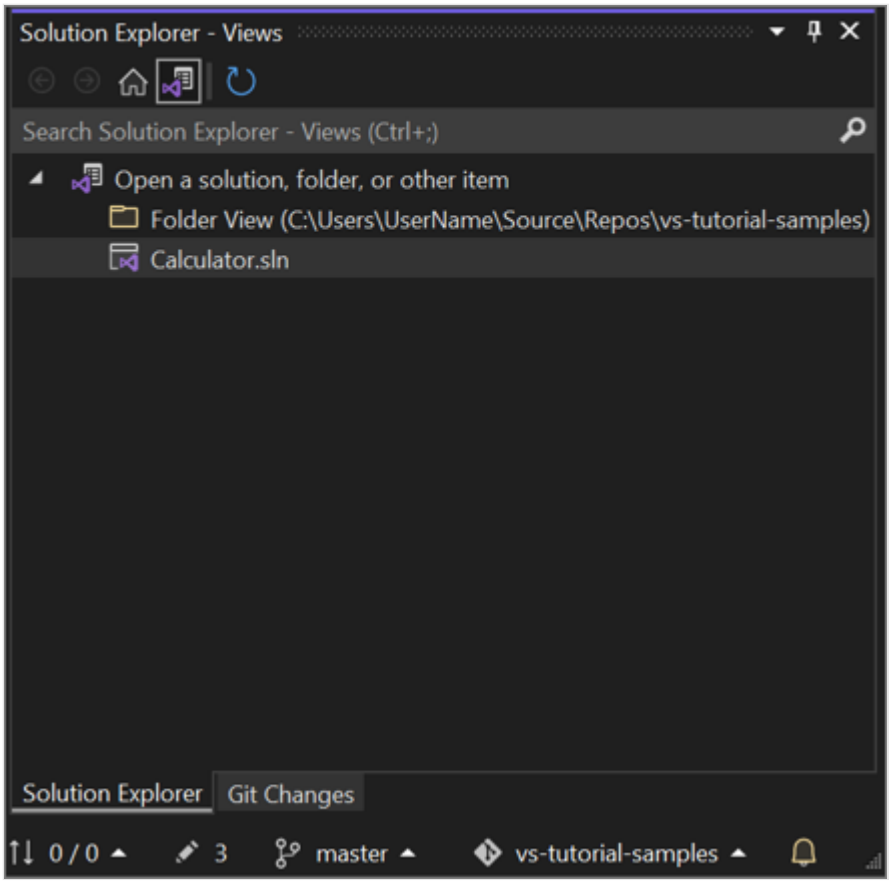


4. 如果尚未登录，系统可能会提示你登录到 Visual Studio 或 GitHub 帐户。

> **提示**
>
> 有关登录 Visual Studio 的详细信息，请参阅**登录 Visual Studio** 页。 有关如何使用 GitHub 帐户进行登录的具体信息，请参阅**在 Visual Studio 中使用 GitHub 帐户**页。 如果你收到信任通知，想要了解有关它的详细信息，请参阅**配置文件和文件夹的信任设置**页。

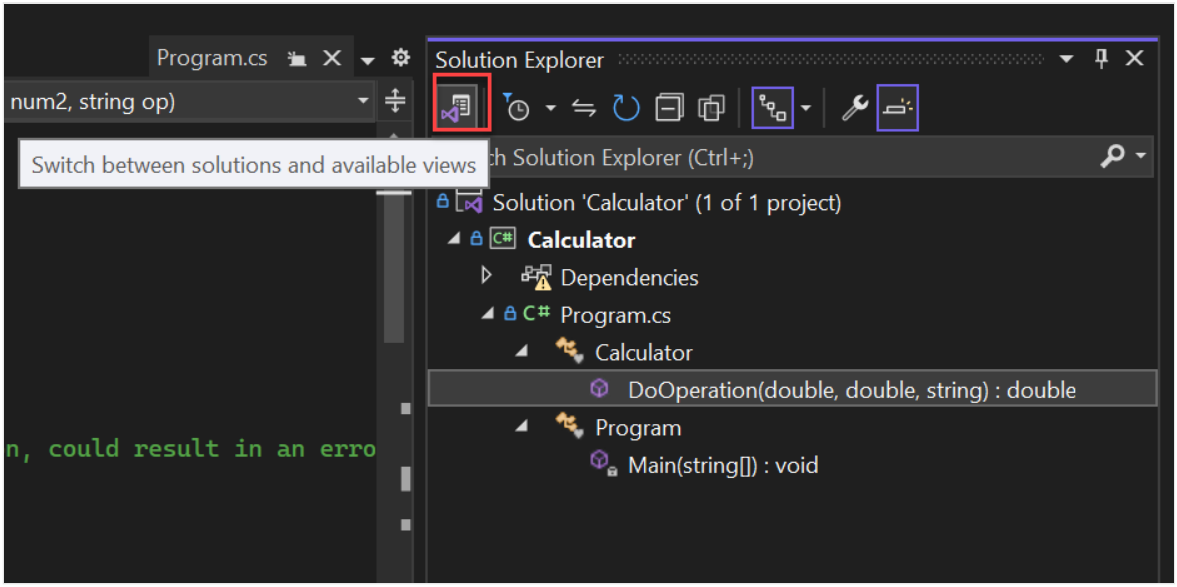
解决方案资源管理器中的的视图文件

1. 接下来，Visual Studio 会通过使用解决方案资源管理器中的“文件夹视图”，从存储库加载解决方案。



可以在“解决方案视图”中查看某个解决方案，方法是双击其 .sln 文件。

也可以选择“切换视图”按钮，然后选择“Program.cs”来查看解决方案的代码。



提示

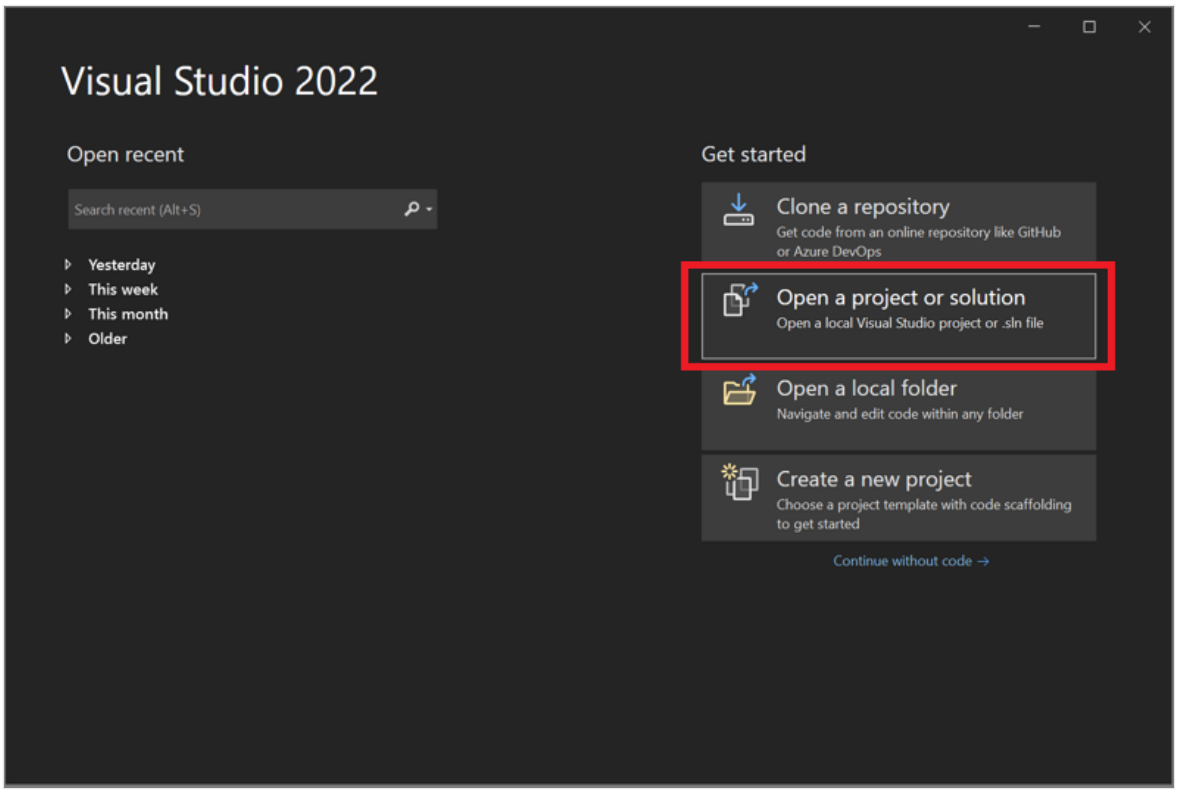
可以通过 **Git** 菜单将默认“文件夹视图”更改为“解决方案视图”。 选择“设置”>“源代码管理”>“Git 全局设置”>“打开 Git 存储库时自动加载解决方案”，以执行该操作。

在本地打开之前克隆的 GitHub 存储库中的项目

1. 打开 Visual Studio。

2. 在起始窗口中，选择“打开项目或解决方案”。

   Visual Studio 将打开文件资源管理器的一个实例，你可以在其中浏览到你的解决方案或项目，然后选择它以将其打开。



> **提示**
>
> 如果最近打开过该项目或解决方案，可从“打开最近使用的文件”部分中将其选中，快速地再次将其打开。

   开始编码！

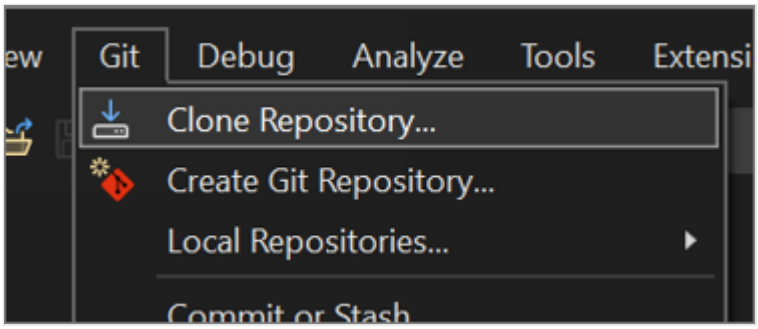
使用 IDE

还可以在 Visual Studio IDE 中使用“Git”菜单或“选择存储库”控件，与存储库的文件夹和文件进行交互。

操作方法如下。

克隆存储库并打开项目

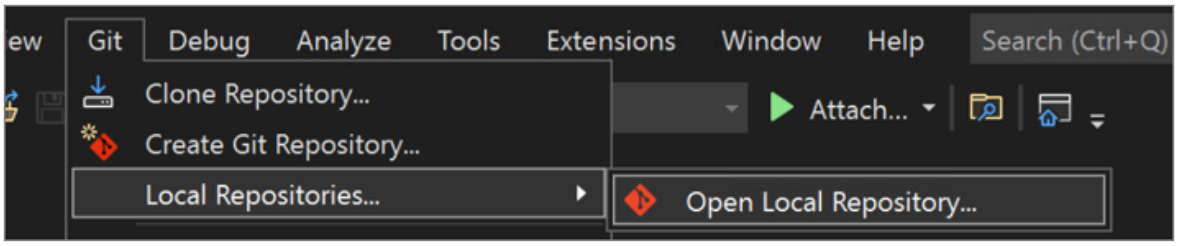
1. 在 Visual Studio IDE 中选择“Git”菜单，然后选择“克隆存储库”。



2. 按照提示连接到包含要查找的文件的 Git 存储库。

打开本地文件夹和文件

1. 在 Visual Studio IDE 中依次选择“Git”菜单、“本地存储库”、“打开本地存储库”。



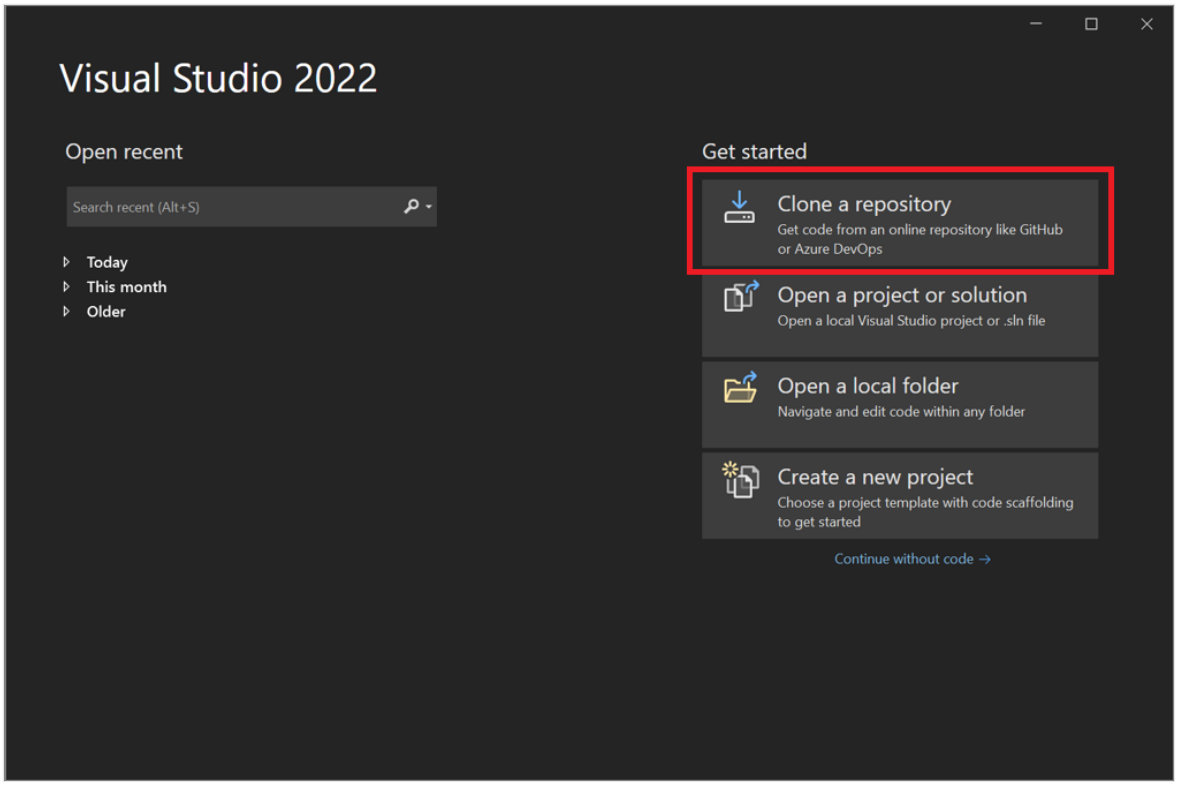
2. 按照提示连接到包含要查找的文件的 Git 存储库。

浏览到 Azure DevOps 存储库

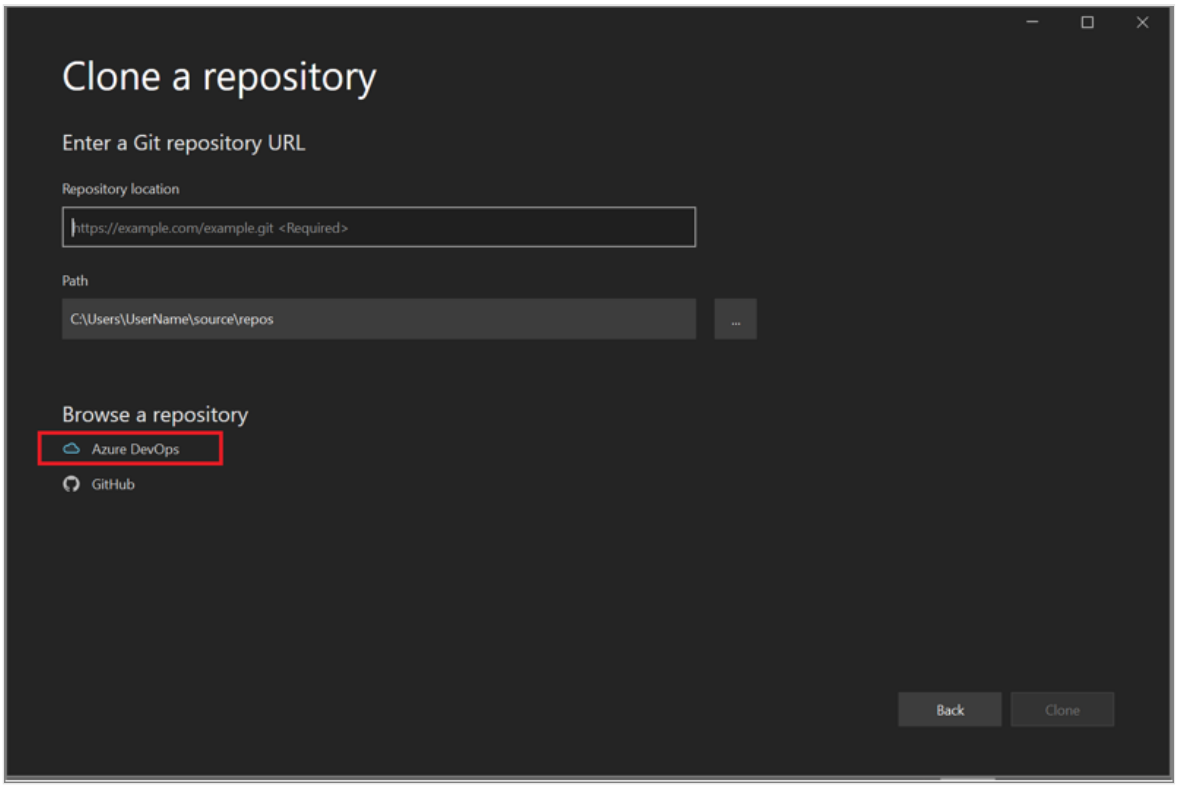
下面介绍如何使用 Visual Studio 浏览到并克隆 Azure DevOps 存储库。

1. 打开 Visual Studio。

2. 在起始窗口中，选择“克隆存储库”。



3. 在“浏览存储库”部分中，选择“Azure DevOps” 。



4. 按照提示克隆要查找的文件所在的 Azure DevOps 存储库，然后打开你的项目。

后续步骤

如果需要，可以深入了解以下特定于语言的任何教程：

- Visual Studio 教程 | C#

- Visual Studio 教程 | Visual Basic
- Visual Studio 教程 | C++
- Visual Studio 教程 | Python
- Visual Studio 教程 | JavaScript、TypeScript 和 Node.js

另请参阅

Visual Studio 版本控制文档

了解如何将代码编辑器用于 C#

项目 • 2023/06/19

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

在这个 10 分钟的 Visual Studio 代码编辑器简介中，我们会向文件添加代码，了解 Visual Studio 编写、导航和了解 C# 代码的简便方法。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

本文假定你已熟悉 C#。 如果不熟悉，建议先看看 Visual Studio 中的 C# 和 ASP.NET Core 入门教程。

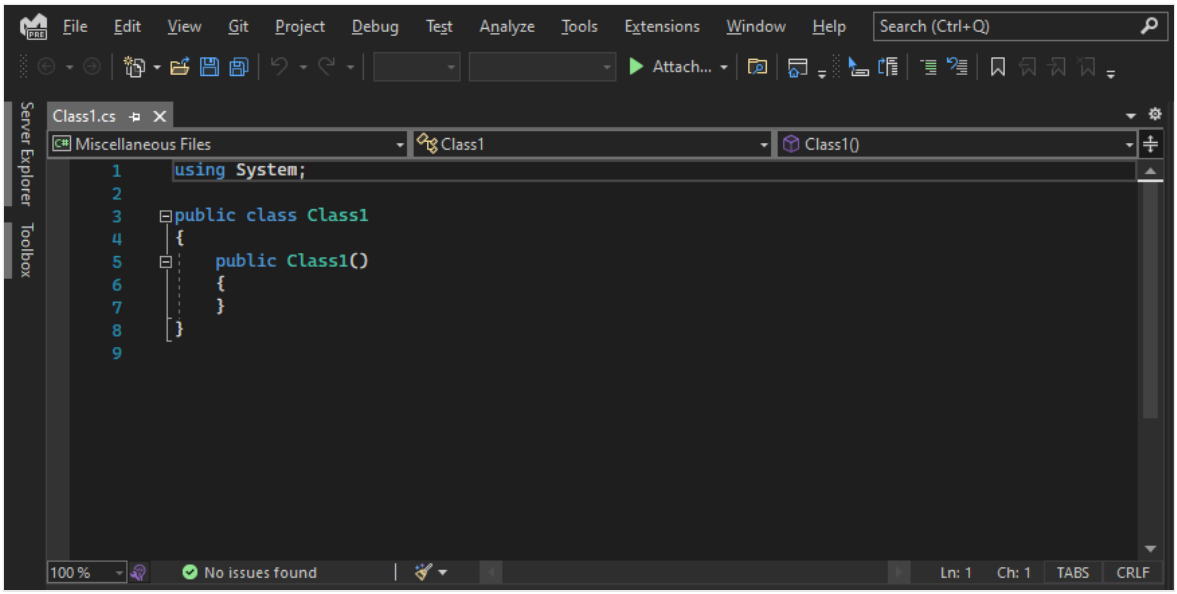
> **提示**
>
> 若要继续执行本文中的操作，请确保为 Visual Studio 选择了 C# 设置。 有关为集成开发环境 (IDE) 选择设置的信息，请参阅**选择环境设置**。

创建新代码文件

先创建一个新文件并向其添加一些代码。

1. 打开 Visual Studio。 按 Esc 或选择“开始”窗口中的“继续但无需代码”以打开开发环境。
2. 在菜单栏上的“文件”菜单中，选择“新建”>“文件”，或按 Ctrl+N 。
3. 在“新建文件”对话框的“常规”类别中，选择“Visual C# 类”，然后选择“打开” 。

   编辑器中将打开主干为 C# 类的新文件。 无需创建完整的 Visual Studio 项目来获取代码编辑器提供的某些益处，仅需一个代码文件即可。



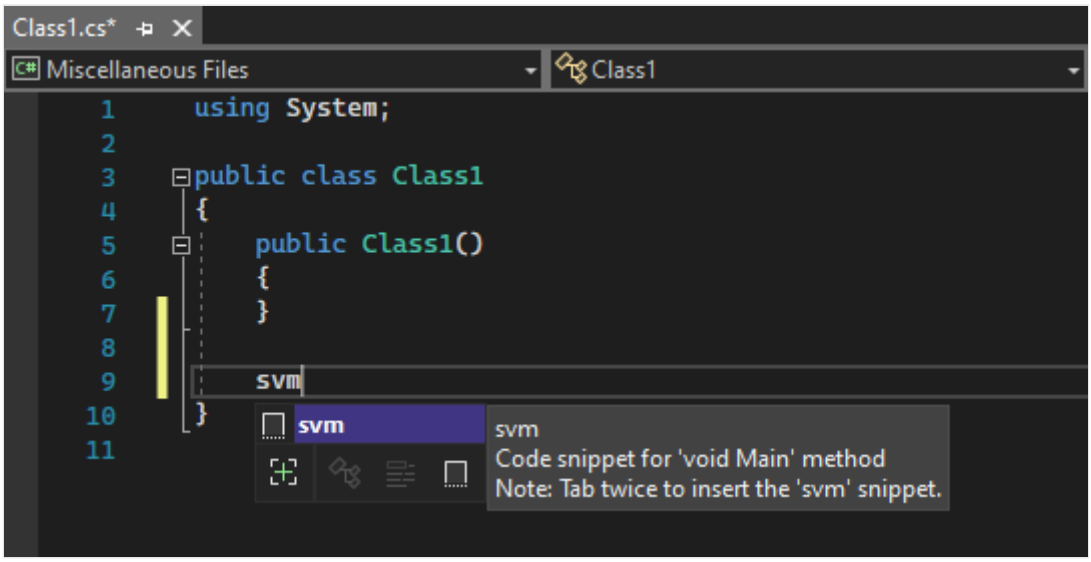
使用代码片段

Visual Studio 提供了实用的代码片段，可用于快速方便地生成常用代码块。 代码片段可用于不同编程语言，包括 C#、Visual Basic 和 C++。

我们将 C# `void Main` 代码片段添加到文件。

1. 将光标停在文件中最后的结束括号 `}` 的上方，并键入字符 `svm`。 `svm` 表示 `static void Main`，如果还不了解此含义，请不要担心。

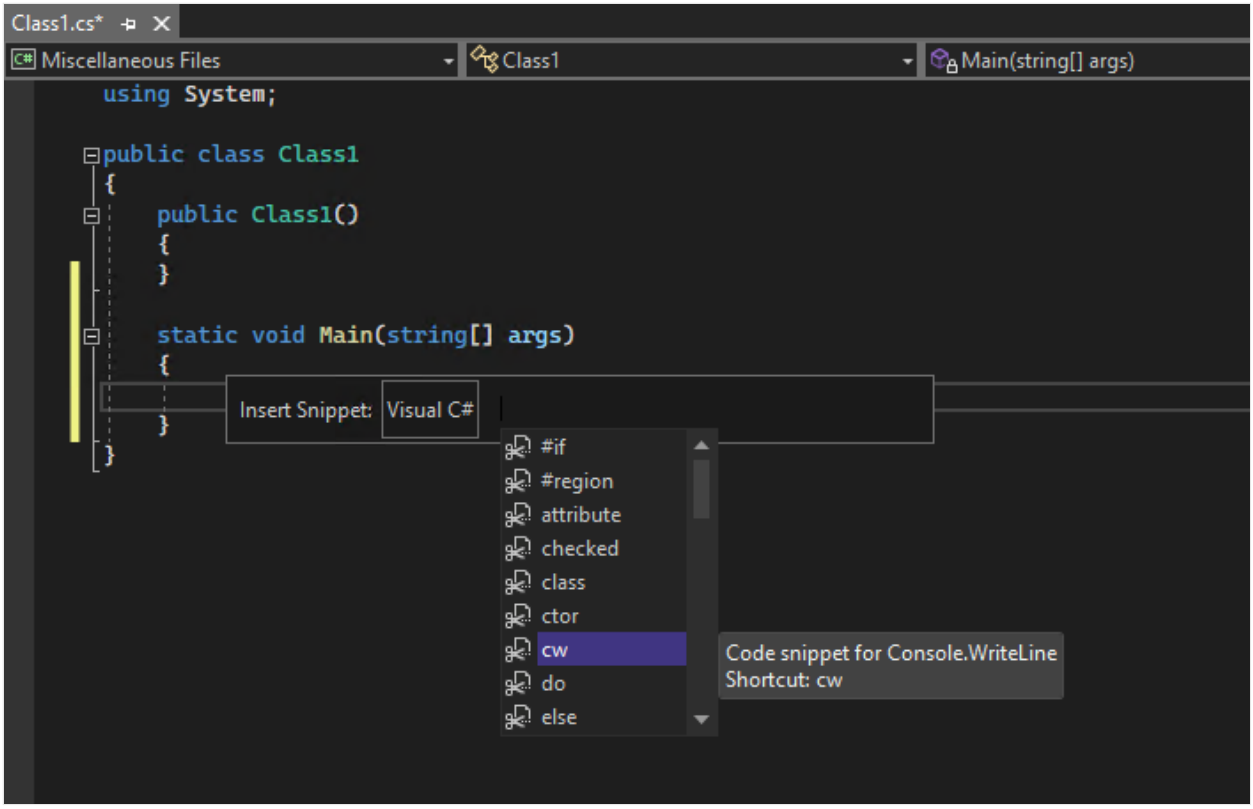
   随即将出现一个弹出对话框，其中包含有关 `svm` 代码片段的信息。



2. 按 Tab 两次，插入代码片段。

   你会看到 `static void Main()` 方法签名被添加到文件。 Main() 方法是 C# 应用程序的入口点。

对于不同编程语言，可用的代码片段不同。 依次选择“编辑”>“IntelliSense”>“插入代码片段”，或按 Ctrl+K、Ctrl+X，然后选择编程语言的文件夹，即可查看该语言的可用代码片段 。 对于 C#，代码片段列表如下所示：



该列表包含用于创建类、构造函数、for 循环、if 或 switch 语句等的代码片段。

为代码添加注释

工具栏是 Visual Studio 菜单栏下的一行按钮，有助于提高编码效率。 例如，可以切换到 IntelliSense 完成模式、增加或减少缩进，或者注释掉不想编译的代码。

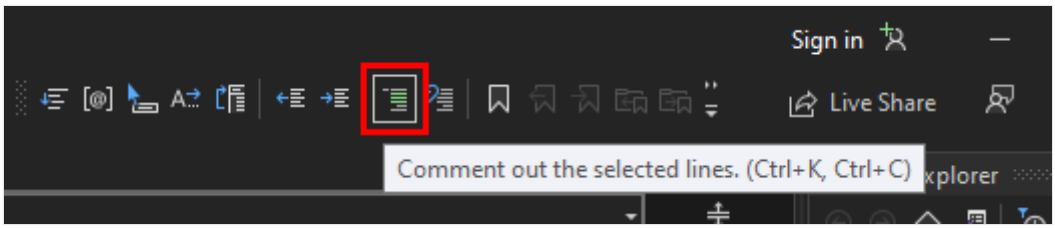
让我们注释掉一些代码。

1. 将以下代码粘贴到 `Main()` 方法主体中。

```C#
// someWords is a string array.
string[] someWords = {
    "the",
    "quick",
    "brown",
    "fox",
    "jumps"
```

```
};

string[] moreWords = {
    "over",
    "the",
    "lazy",
    "dog"
};

// Alphabetically sort the words.
IEnumerable<string> query = from word in someWords
                            orderby word
                            select word;
```

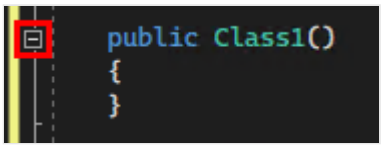
2. 我们现在没有使用 `moreWords` 变量，但稍后可能会用到，所以我们不想删除它。 那我们就来注释掉这些行。 选择整个 `moreWords` 定义直到结束分号，然后选择工具栏上的"注释掉选定行"。 如果想要使用键盘，请按 Ctrl+E, Ctrl+C 。



C# 注释字符 `//` 添加到了每个所选行的开始处，从而为代码添加注释。

折叠代码块

我们不想看到生成的 `Class1` 的空构造函数，所以为了让代码更整洁，我们将其折叠。 在构造函数第一行的边距中选择内部带有减号的小灰色框。 如果首选使用键盘，也可将光标置于构造函数代码中的任意位置，然后按 Ctrl+M、Ctrl+M 。



代码块折叠到第一行，后跟省略号 (`...`)。 若要再次展开代码块，请选择现在带有加号的相同灰色框，或者再次按 Ctrl+M，Ctrl+M 。 此功能被称为大纲显示，在折叠长方法或整个类时特别有用。

查看符号定义

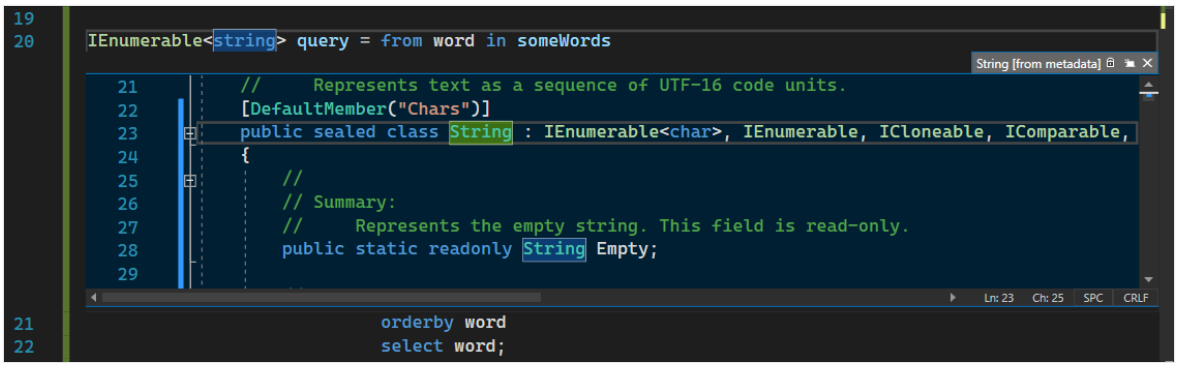
通过 Visual Studio 编辑器，可轻松查看类型、方法或变量的定义。 一种方法是转到包含定义的任何文件，例如通过选择"转到定义"或按"F12"，转到引用符号的任何位置 。 使用

“速览定义”速度更快，不会干扰你处理代码。

我们来快速查看一下 `string` 类型的定义。

1. 右键单击出现的任意 `string`，然后选择内容菜单上的“速览定义”。 或者，按 Alt+F12 。

   此时会出现一个弹出窗口，其中包含 `String` 类的定义。 可在弹出窗口中滚动，甚至还可从速览的代码中查看另一类型的定义。



2. 选择弹出窗口右上方的“x”小框，关闭“速览定义”窗口。

使用 IntelliSense 完成单词

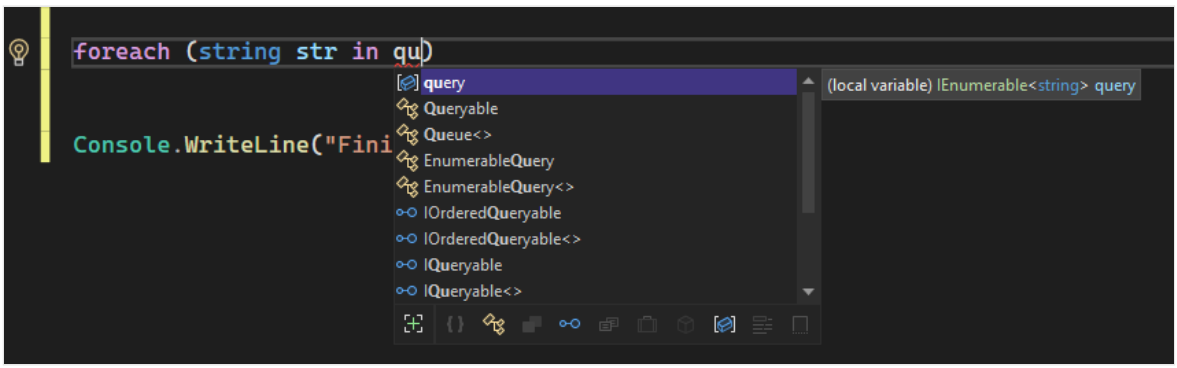
编写代码时，IntelliSense 是非常宝贵的资源。 它可显示某个类型的可用成员信息，或某个方法不同重载的参数详情。 还可用于完成单词，从而在输入大量字符后消除字符带来的歧义。

添加代码行，将有序字符串呈现到控制台窗口，这是程序输出的标准位置。

1. 在 `query` 变量下，开始键入以下代码：

```
C#

foreach (string str in qu
```

   此时会看到一个 IntelliSense 弹出项，其中包含有关 `query` 符号的相关信息。



2. 若要使用 IntelliSense 文字自动完成插入单词 `query` 的剩余部分，请按 Tab。

3. 完成后，代码块如以下代码所示。 你可以通过输入 `cw`，然后按 Tab 两次来生成 `Console.WriteLine` 语句，从而进一步练习代码片段。

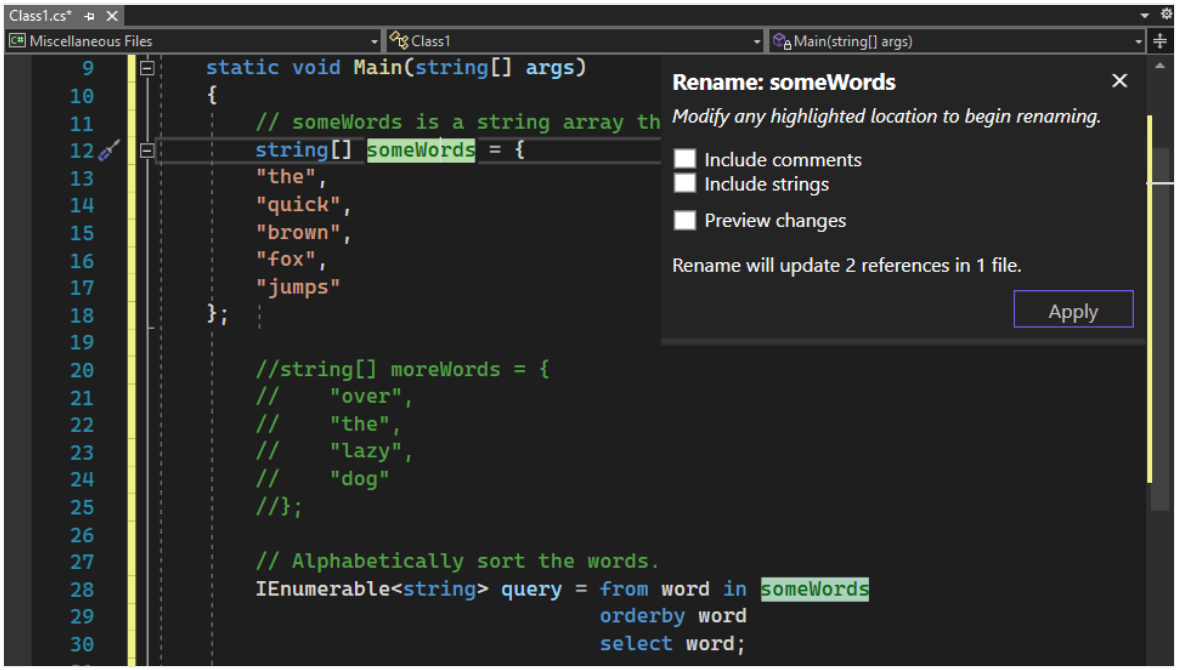
```
C#

foreach (string str in query)
{
   Console.WriteLine(str);
}
```

重构名称

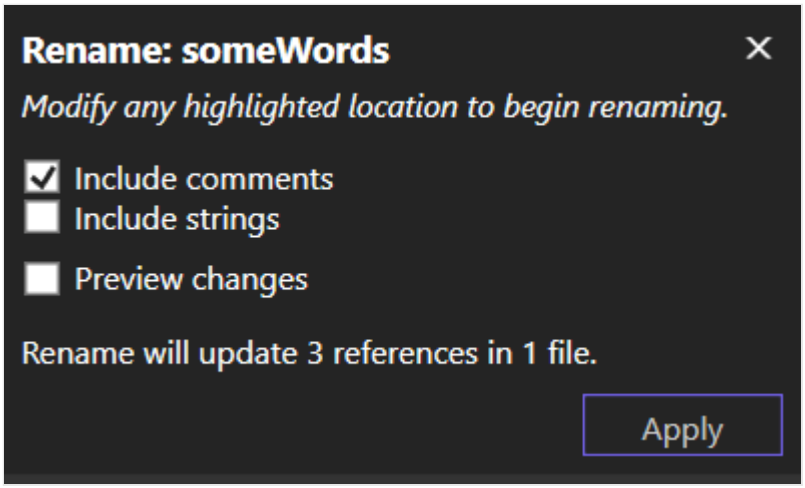
没有谁能一次就得到正确的代码，代码中可能必须要更改的一项内容是变量或方法的名称。 我们来试试 Visual Studio 的重构功能，将 `someWords` 变量重命名为 `unsortedWords`。

1. 将光标置于 `someWords` 变量的定义上，然后从右键菜单或上下文菜单中选择“重命名”，或按 F2。

   此时编辑器右上角会出现一个“重命名”对话框。



2. 输入所需名称“unsortedWords”。 你会看到查询中对 `query` 赋值语句中 `unsortedWords` 的引用也会自动重命名。 在按 Enter 前，请在“重命名”弹出框中选中“包含注释”复选框 。



3. 按 Enter，或在“重命名组”对话框中选择“应用”。

   代码中出现的两处 `someWords` 均被重命名，代码注释中的文本 `someWords` 也被重命名。

后续步骤

了解项目和解决方案

请参阅

- 代码片段
- 导航代码
- 大纲显示
- 转到定义和速览定义
- 重构
- 使用 IntelliSense

在 Visual Studio 中编译和生成

项目 • 2024/01/18

若要初步了解如何在 IDE 中进行生成，请参阅演练：生成应用程序。

可以使用以下任何方法来生成应用程序：Visual Studio IDE、MSBuild 命令行工具和 Azure Pipelines：

展开表

生成方法	优点
IDE	- 立即创建生成并在调试程序中对其进行测试。 - 运行 C++ 和 C# 项目的多处理器生成。 - 自定义生成系统的不同方面。
CMake	- 使用 CMake 工具生成 C++ 项目 -跨 Linux 和 Windows 平台使用同一生成系统。
MSBuild 命令行	- 在无需安装 Visual Studio 的情况下生成项目。 - 运行所有项目类型的多处理器生成。 - 自定义生成系统的大多数区域。
Azure Pipelines	- 自动执行生成过程作为持续集成/持续交付管道的一部分。 - 将自动测试应用于每个生成。 - 为生成过程采用几乎无限的基于云的资源。 - 修改生成工作流，并创建生成活动以执行深层的自定义任务。

本节中的文档将详细介绍基于 IDE 的生成过程。 有关其他方法的详细信息，请分别参阅 CMake、MSBuild 和 Azure Pipelines。

从 IDE 生成

创建项目时，Visual Studio 创建了该项目的默认生成配置和包含该项目的解决方案。 这些配置定义如何生成和部署解决方案和项目。 尤其对目标平台（如 Windows 或 Linux）和生成类型（如调试或发布）而言，项目配置必须唯一。 但是，你可以根据喜好编辑这些配置，也可以根据需要创建自己的配置。

若要初步了解如何在 IDE 中进行生成，请参阅演练：生成应用程序。

接下来，请参阅在 Visual Studio 中生成和清理项目和解决方案，了解可以对过程进行哪些不同的自定义设置。 自定义包括更改输出目录、指定自定义生成事件、管理项目依赖项、管理生成日志文件以及禁止显示编译器警告。

你还可以浏览各种其他任务：

- 了解生成配置
- 将项目配置为面向平台
- 管理项目和解决方案属性。
- 指定 C# 和 Visual Basic 中的生成事件。
- 设置生成选项
- 并行生成多个项目。

相关内容

- 生成（编译）网站项目
- Visual Studio 中的 CMake 项目

反馈

此页面是否有帮助? 是 否

提供产品反馈 | 询问社区

教程：了解如何使用 Visual Studio 调试 C# 代码

项目 • 2023/11/30

适用范围： Visual Studio Visual Studio for Mac Visual Studio Code

本文通过分步演练介绍了 Visual Studio 调试器的功能。 如果需要更加深入地了解调试器功能，请参阅初探调试器。 当你调试应用时，通常意味着运行附带有调试器的应用程序。 执行此任务时，调试器在运行过程中可提供许多方法让你查看代码的情况。 你可以逐步浏览代码、查看变量中存储的值、设置对变量的监视以查看值何时改变、检查代码的执行路径、查看代码分支是否正在运行等等。 如果这是你第一次在本练习中尝试调试代码，建议在浏览本文之前阅读零基础调试。

虽然演示应用为 C#，但大多数功能都适用于 C++、Visual Basic、F#、Python、JavaScript 和 Visual Studio 支持的其他语言（F# 不支持“编辑并继续”。F# 和 JavaScript 不支持“自动”窗口）。 屏幕截图为 C#。

在本教程中，你将：

- ✓ 启动调试器并命中断点。
- ✓ 了解在调试器中逐步执行代码的命令
- ✓ 检查数据提示和调试器窗口中的变量
- ✓ 检查调用堆栈

先决条件

须安装 Visual Studio 2022 且具有“.NET 桌面开发”工作负载。

如果尚未安装 Visual Studio，请转到 Visual Studio 下载 页免费安装。

如果你已有 Visual Studio 但未安装“.NET 桌面开发”工作负载，请转到“工具”>“获取工具和功能...”，随即会启动 Visual Studio 安装程序 。 在 Visual Studio 安装程序中，选择“.NET 桌面开发”工作负载，然后选择“修改” 。

创建项目

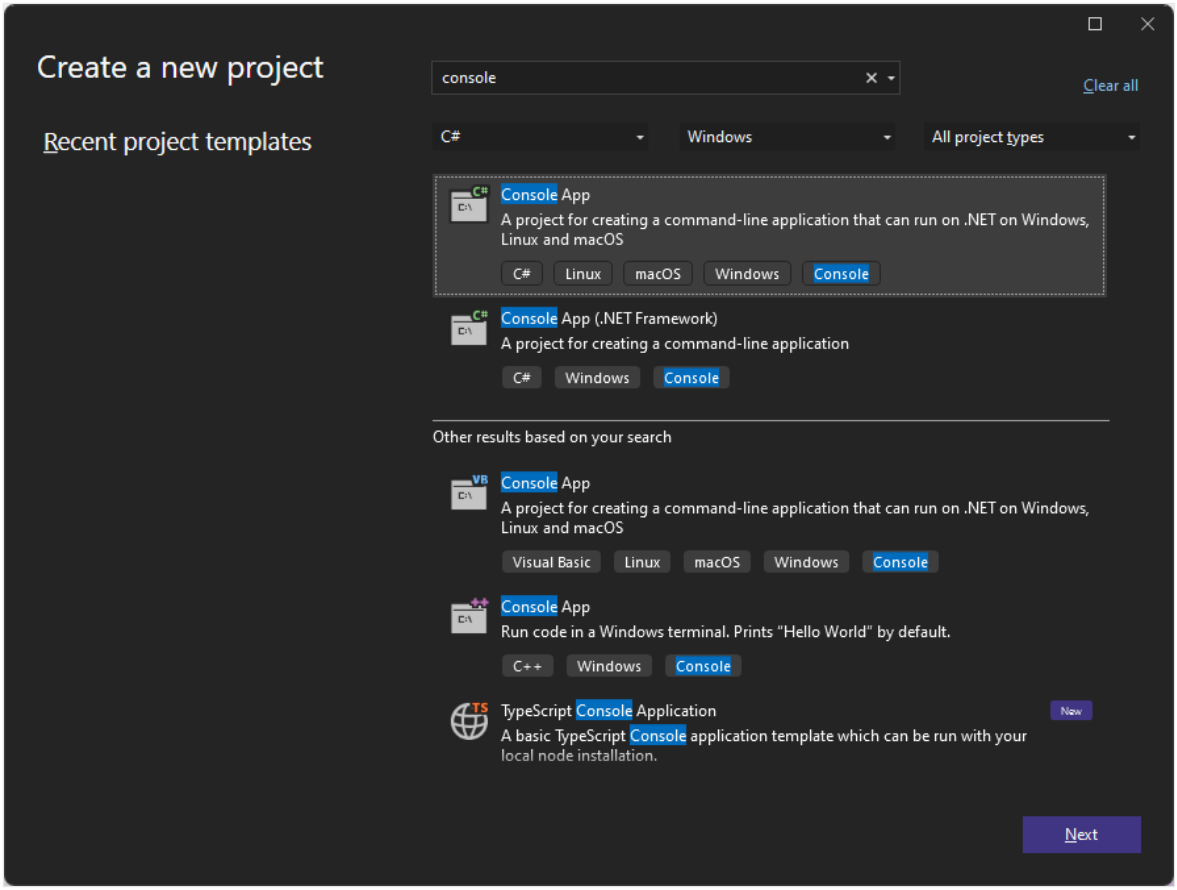
首先，创建一个 .NET Core 控制台应用程序项目。 项目类型随附了所需的全部模板文件，无需添加任何内容！

1. 打开 Visual Studio。 如果“开始”窗口未打开，请选择“文件”>“启动窗口”。

2. 在“开始”窗口中，选择“创建新项目”。

3. 在“创建新项目”窗口的搜索框中输入“控制台”。 接下来，从“语言”列表中选择 C#，然后从“平台”列表中选择 Windows 。

   应用语言和平台筛选器之后，选择“控制台应用”模板，然后选择“下一步”。



> ⓘ **备注**
>
> 如果未看到“控制台应用”模板，则可以通过“创建新项目”窗口安装该模板。 在“找不到所需内容?”消息中，选择“安装更多工具和功能”链接 。 然后在 Visual Studio 安装程序中，选择“.NET 桌面开发”工作负载。

4. 在“配置新项目”窗口中，在“项目名称”框中输入“GetStartedDebugging”。 然后选择**下一步**。

5. 在“其他信息”窗口中，确保在“框架”下拉菜单中选择“.NET 8.0”，然后选择“创建”。

此时，Visual Studio 将打开新项目。

创建应用程序

在 Program.cs 中，将所有默认代码替换为以下代码：

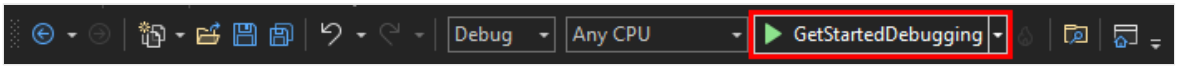
C#

```
using System;

class ArrayExample
{
   static void Main()
   {
      char[] letters = { 'f', 'r', 'e', 'd', ' ', 's', 'm', 'i', 't', 'h'};
      string name = "";
      int[] a = new int[10];
      for (int i = 0; i < letters.Length; i++)
      {
         name += letters[i];
         a[i] = i + 1;
         SendMessage(name, a[i]);
      }
      Console.ReadKey();
   }

   static void SendMessage(string name, int msg)
   {
      Console.WriteLine("Hello, " + name + "! Count to " + msg);
   }
}
```

启动调试器！

此处我们主要使用的是键盘快捷方式，因为这样可以快速执行调试器命令。 等效的命令（例如工具栏或菜单命令）也会注明。

1. 若要启动调试器，请按 F5，或者在“标准”工具栏中选择“调试目标”按钮，或者在“调试”工具栏中选择“开始调试”按钮，或者从菜单栏中选择“调试”>“开始调试”。

   按 F5 会启动应用，并且调试器会附加到应用进程。 由于我们尚未执行任何特殊操作来检查代码，因此应用将一直运行到完成为止，并且你会看到控制台输出。

   Windows 命令提示符

   ```
   Hello, f! Count to 1
   Hello, fr! Count to 2
   Hello, fre! Count to 3
   Hello, fred! Count to 4
   Hello, fred ! Count to 5
   Hello, fred s! Count to 6
   Hello, fred sm! Count to 7
   Hello, fred smi! Count to 8
   ```

```
Hello, fred smit! Count to 9
Hello, fred smith! Count to 10
```

2. 若要停止调试器，请按 Shift+F5，或者在“调试”工具栏中选择“停止调试”按钮，或者从菜单栏中选择“调试”>“停止调试”。

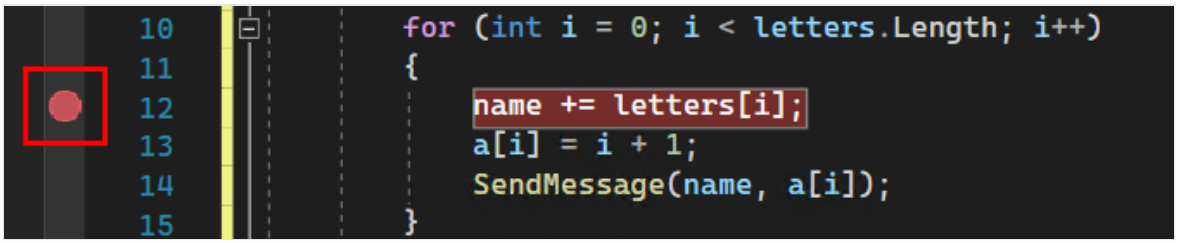
3. 在控制台窗口中，按任意键关闭控制台窗口。

设置断点并启动调试器

1. 在 `Main` 函数的 `for` 循环中，通过单击以下代码行的左边距来设置断点：

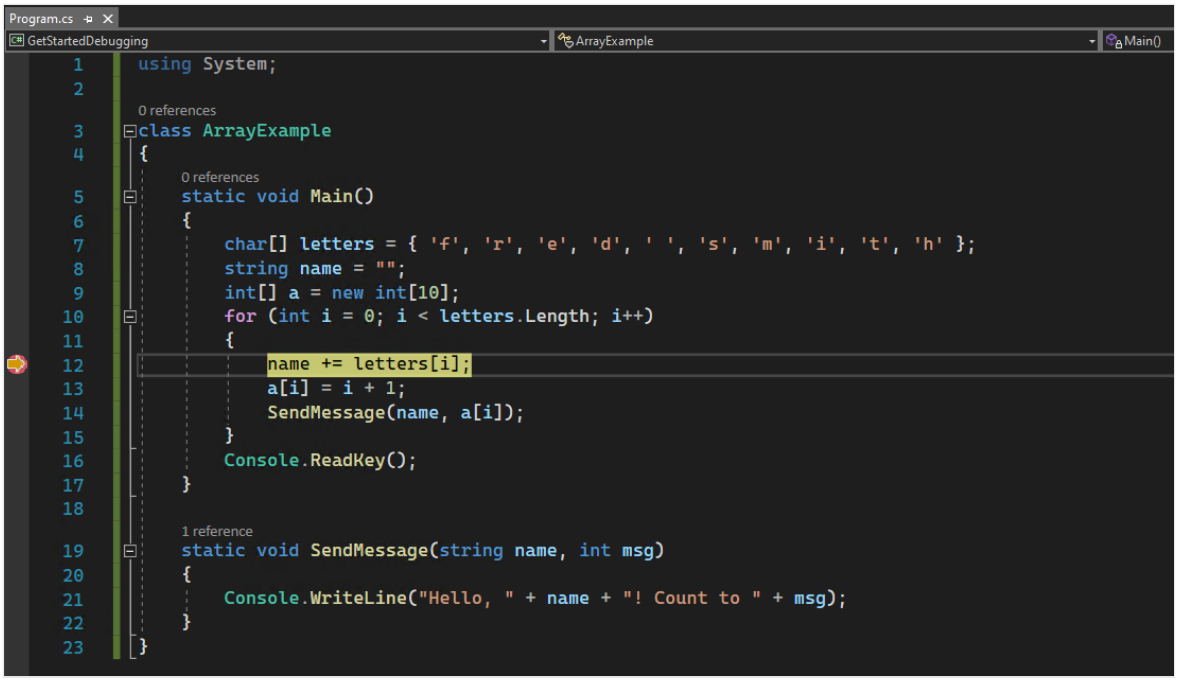
   `name += letters[i];`

   设置断点的位置会出现一个红色圆圈。



   断点是执行可靠调试所不可或缺的一项功能。 你可以设置断点，以便 Visual Studio 在你设置的断点处暂停正在运行的代码，这样你就可以查看变量值或内存行为，或者确定代码的分支是否已运行。

2. 若要开始调试，请按 F5，或者在“标准”工具栏中选择“调试目标”按钮，或者在“调试”工具栏中选择“开始调试”按钮，或者从菜单栏中选择“调试”>“开始调试” 。 应用随即启动，调试器将运行到你设置了断点的代码行。



黄色箭头指向调试器暂停时所在的语句。 应用执行将在同一位置暂停，此处的语句尚未执行。

当应用未运行时，按 F5 会启动调试器。调试器将持续运行该应用，直至到达第一个断点。 如果应用在某个断点处暂停，则按 F5 会继续运行该应用，直至到达下一个断点。

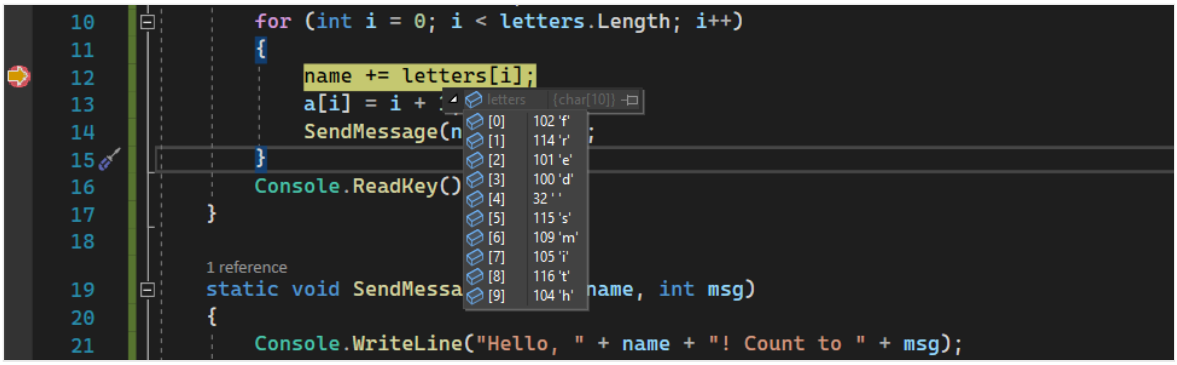
如果你知道自己要详细检查的代码行或代码段，则断点功能非常有用。 有关可设置的各种断点（例如条件断点）的详细信息，请参阅使用断点。

使用数据提示浏览代码并检查数据

1. 当代码执行在 `name += letters[i]` 语句处暂停时，将鼠标悬停在 `letters` 变量上以查看显示数组大小和元素类型 `char[10]` 的数据提示。

   > **备注**
   >
   > 调试器最有用的功能之一是能够检查变量。 通常，在尝试调试某个问题时，你会试图确定变量是否具有你期望它们在特定时间具有的值。 查看数据提示是进行这种检查的好办法。

2. 展开 `letters` 变量以查看其所有数组元素以及这些元素的值。

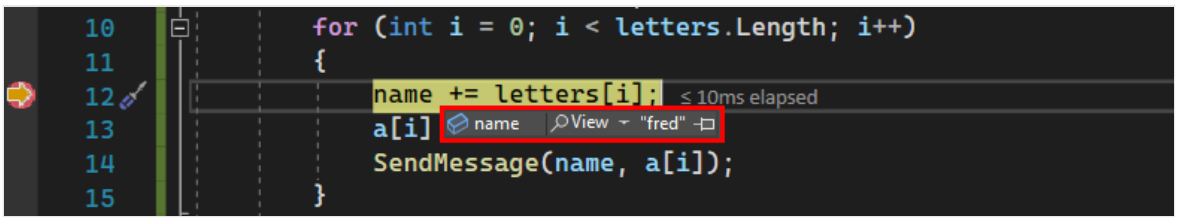


3. 将鼠标悬停在 `name` 变量上以查看其当前值（一个空字符串）。

4. 若要使调试器前进到下一条语句，请按 F10，或者在“调试”工具栏中选择“单步跳过”按钮，或者从菜单栏中选择“调试”>“单步跳过” 。 再按 F10 两次以跳过 `SendMessage` 方法调用。

   按 F10 会使调试器前进，而不会单步执行函数或方法，不过，其代码仍会执行。 这样，就跳过了调试我们暂时不需要关注的 `SendMessage` 方法中的代码。

5. 若要迭代 `for` 循环几次，请反复按 F10。 在每次循环迭代期间，请在断点处暂停，然后将鼠标悬停在 `name` 变量上以在数据提示中检查其值。



   变量的值随 `for` 循环的每次迭代而更改，显示的值依次为 `f`、`fr`、`fre`，依此类推。 若要使调试器在循环中更快前进，请按 F5，这样就会前进到断点而不是下一条语句。

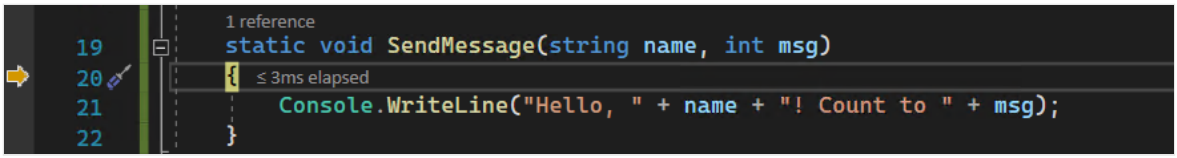
6. 当代码执行在 `Main` 方法的 `for` 循环中暂停时，按 F11，或者从“调试”工具栏中选择“单步执行”按钮，或者从菜单栏中选择“调试”>“单步执行”，直至到达 `SendMessage` 方法调用。

   应在此代码行处暂停调试器：

   ```
   SendMessage(name, a[i]);
   ```

7. 若要单步执行 `SendMessage` 方法，请再次按 F11。

   黄色指针会前进到 `SendMessage` 方法。



按 F11 可帮助你更深入地检查代码的执行流。 若要单步执行方法调用中的方法，请按 F11。 默认情况下，调试器将跳过非用户方法的单步执行。 若要了解如何调试非用户代码，请参阅仅我的代码。

调试完 `SendMessage` 方法后，可以返回到 `main` 方法的 `for` 循环。

8. 若要退出 `SendMessage` 方法，请按 Shift+F11，或者在“调试”工具栏中选择“单步跳出”按钮，或者从菜单栏中选择“调试”>“单步跳出”。

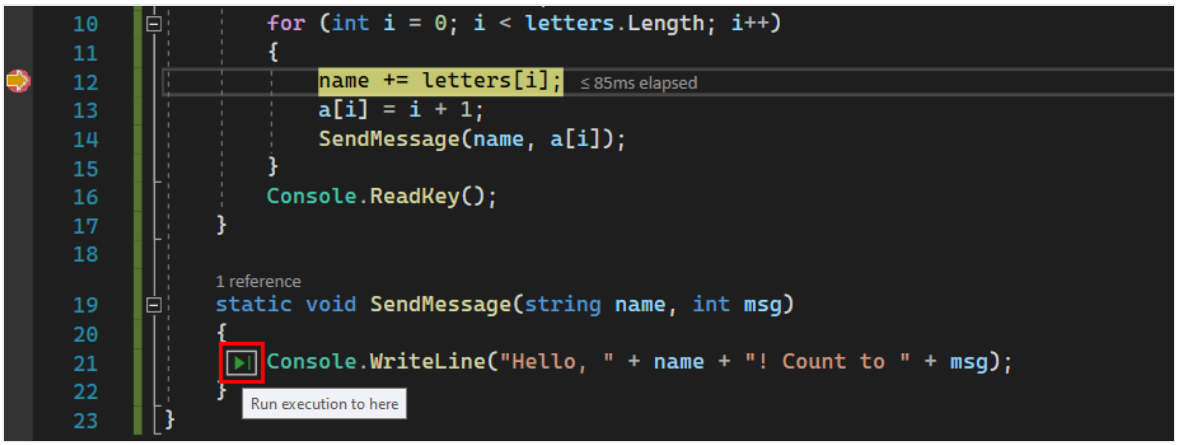
   “单步跳出”将恢复应用执行并使调试器前进，直到当前方法或函数返回。

   你将在 `Main` 方法的 `for` 循环中再次看到黄色指针，该指针暂停在 `SendMessage` 方法调用处。 有关在代码中进行移动的不同方法的详细信息，请参阅浏览调试器中的代码。

使用“运行时单击”导航代码

1. 按 F5 再次前进到断点。

2. 在代码编辑器中，将鼠标悬停在 `SendMessage` 方法中的 `Console.WriteLine` 方法调用上，直到出现“运行到单击处”按钮。 按钮的工具提示显示“将执行运行到此处”。



3. 选择“运行到单击处”按钮。 或者，将光标置于 `Console.WriteLine` 语句上，然后按 Ctrl+F10。 或者，右键单击 `Console.WriteLine` 方法调用，然后从上下文菜单中选择“运行到光标处”。

   调试器会前进到 `Console.WriteLine` 方法调用。

使用“运行到单击处”按钮类似于设置临时断点，在已打开的文件的应用代码可见区域中，可以快速方便地使用这种方法。

快速重启应用

若要在调试器中从头开始重新运行应用，请按 Ctrl+Shift+F5，或者在“调试”工具栏中选择“重启”按钮，或者从菜单栏中选择“调试”>“重启”。

“重启”将通过一个步骤停止调试器并重启调试器。 调试器重启时，将运行到第一个断点（先前在 `for` 循环中设置的断点），然后暂停。

使用“自动”和“局部变量”窗口检查变量

在调试时，“自动变量”和“局部变量”窗口会显示变量值 。 这两个窗口仅在调试会话期间才会显示。 “自动变量”窗口显示调试器所在的当前行和上一行中使用的变量。 “局部变量”窗口显示在局部范围内定义的变量，通常是当前函数或方法。

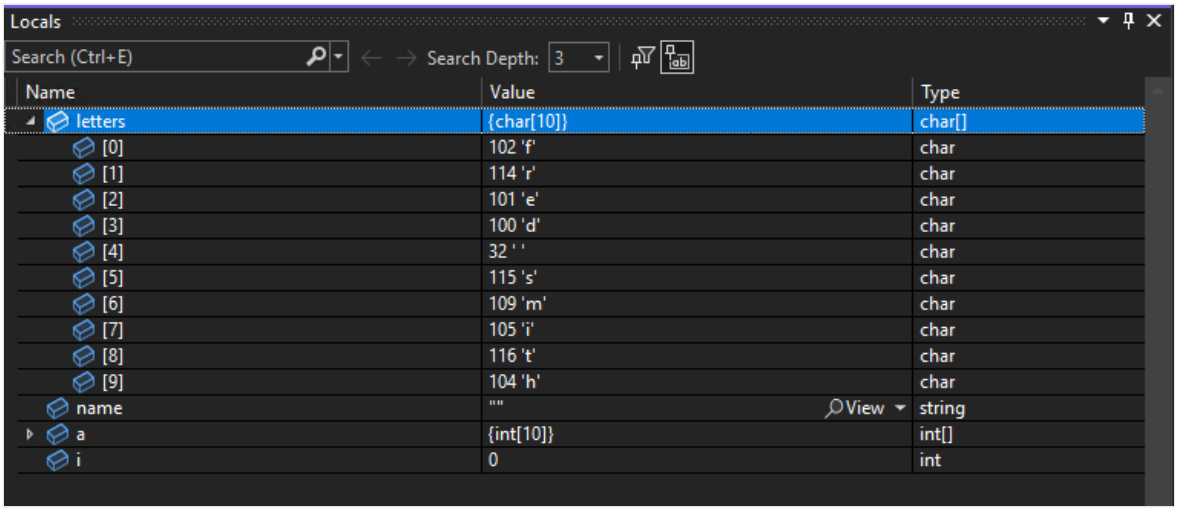
1. 在调试器处于暂停状态时，查看代码编辑器底部的“自动变量”窗口。

   如果“自动变量”窗口已关闭，请按下 Ctrl+D 和 A，或者从菜单栏中选择“调试”>“窗口”>“自动变量”。

2. 在调试器仍处于暂停状态的情况下，在“自动变量”窗口旁边的选项卡中查看“局部变量”窗口 。

   如果“局部变量”窗口已关闭，请按下 Ctrl+D 和 L，或者从菜单栏中选择“调试”>“窗口”>“局部变量”。

3. 在“局部变量”窗口中，展开 `letters` 变量以查看其数组元素以及这些元素的值。



有关“自动变量”和“局部变量”窗口的详细信息，请参阅在“自动变量”和“局部变量”窗口中检查变量 。

设置监视

可以指定你在逐步执行代码时想要观察的变量或表达式 - 将其添加到“监视”窗口即可。

1. 当调试器处于暂停状态时，右键单击 `name` 变量并选择“添加监视”。

   “监视”窗口默认将在代码编辑器的底部打开。

2. 对 `name` 变量设置监视后，接下来请逐步执行代码，以查看 `name` 变量值在每次 `for` 循环迭代中的变化。

   与其他变量窗口不同，“监视”窗口始终显示你正在监视的变量。 超出范围的变量显示为不可用。

有关“监视”窗口的详细信息，请参阅使用“监视”窗口监视变量。

检查调用堆栈

“调用堆栈”显示方法和函数的调用顺序，可帮助你了解应用的执行流。

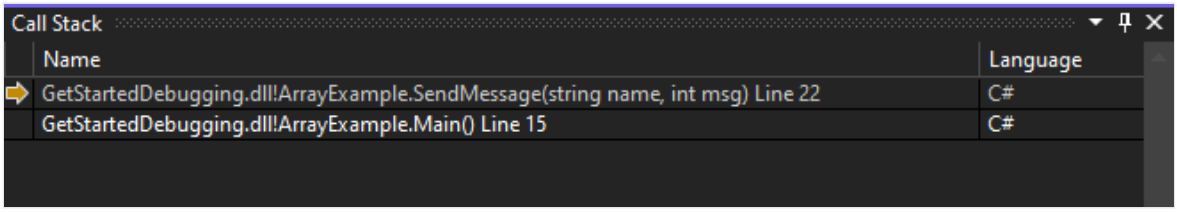
1. 当调试器在 `for` 循环中处于暂停状态时，查看“调用堆栈”窗口，该窗口默认会在代码编辑器的右下窗格中打开。

   如果“调用堆栈”窗口已关闭，请按下 Ctrl+D 和 C，或者从菜单栏中选择“调试”>“窗口”>“调用堆栈”。

   在“调用堆栈”窗口中，你将在当前 `Main` 方法处看到黄色指针。

2. 按 F11 几次，直至看到调试器在 `SendMessage` 方法中暂停。

   “调用堆栈”窗口的第一行显示当前函数，即 `SendMessage` 方法。 第二行显示从 `Main` 方法调用的 `SendMessage` 方法。



> **备注**

"调用堆栈"窗口类似于某些 IDE（例如 Eclipse）中的"调试"透视图。

在"调用堆栈"窗口中，可以双击代码行转到相应的源代码，这会更改调试器正在检查的当前范围。 此操作不会使调试器前进。

还可使用"调用堆栈"窗口中的右键单击菜单执行其他操作 。 例如，可将断点插入到指定的函数中、使用"运行到光标处"使调试器前进，或转到源代码。

有关"调用堆栈"的详细信息，请参阅如何：检查调用堆栈。

更改执行流

在调试时，可以移动执行指针以更改应用流。

1. 当调试器暂停于 `for` 循环中的 `SendMessage` 方法调用处时，按 F11 三次以单步执行 `SendMessage` 方法，并在执行该方法后跳过 `Console.WriteLine` 方法。

   现在，调试器已暂停于 `SendMessage` 方法的最后一个右大括号处。

2. 使用鼠标抓取黄色箭头或执行指针（在左边缘中），然后将指针向上拖动一行。

   调试器现在回到了 `Console.WriteLine` 语句。

3. 按 F11。

   调试器重新运行 `Console.WriteLine` 方法，你将在控制台窗口输出中看到重复的行。

4. 按 F5 继续运行应用。

通过更改执行流，你可以进行测试不同代码执行路径或重新运行代码等操作，而无需重启调试器。

> ⚠ **警告**
>
> 请慎用此功能。 你将在执行指针的工具提示中看到一条警告，其中指出可能会出现意外的后果。 此外，还可能会看到其他警告。 移动执行指针无法将应用程序还原到以前的状态。

有关更改执行流的详细信息，请参阅移动指针以更改执行流。

恭喜你完成本教程！

后续步骤

在本教程中，你已了解了如何启动调试器、逐步执行代码以及检查变量。 你可能想要大致了解调试器功能并获取指向详细信息的链接。

初探调试器

单元测试入门

项目 • 2024/01/19

使用 Visual Studio 定义和运行单元测试，使代码保持正常运行、确保代码覆盖率并在客户之前找到错误和缺陷。 经常运行单元测试，确保代码正常运行。

在本文中，代码使用 C# 和 C++，图例使用 C#，但是概念和特征适用于 .NET 语言、C++、Python、JavaScript 和 TypeScript。

创建单元测试

本节介绍了如何创建单元测试项目。

1. 在 Visual Studio 中，打开要测试的项目。

   出于演示示例单元测试的目的，本文测试一个简单的“Hello World”C# 或名为“Hello World”的 C++ 控制台项目。 此类项目的示例代码如下所示：

   .NET

   C#

   ```
   namespace HelloWorld
   {
      public class Program
      {
         public static void Main()
         {
            Console.WriteLine("Hello World!");
         }
      }
   }
   ```

2. 在“解决方案资源管理器”中，选择解决方案节点。 然后，在顶部菜单栏中，选择“文件”>“添加”>“新项目”。

3. 在新项目对话框中，找到要使用的单元测试项目。

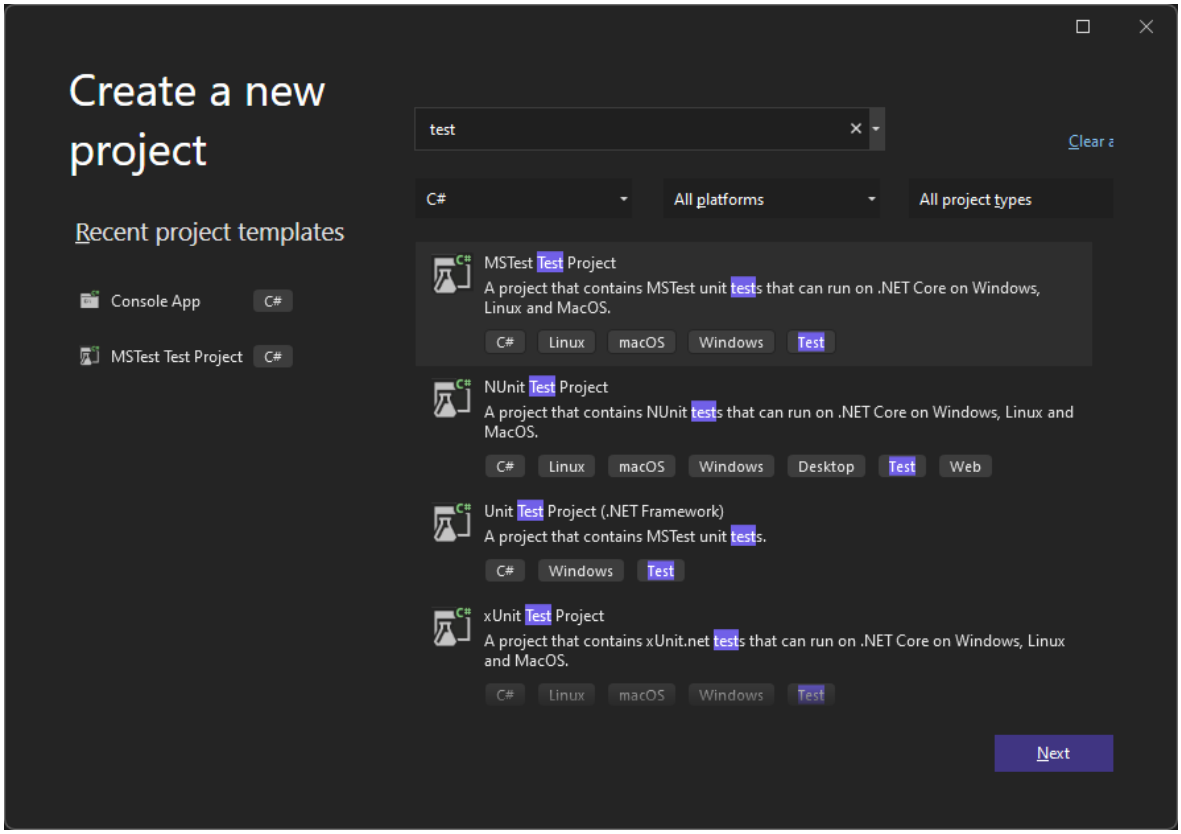
   在搜索框中键入“测试”，找到要使用的测试框架的单元测试项目模板（例如 MSTest (C#) 或本机单元测试项目 (C++)），并选择它。

   从 Visual Studio 2017 14.8 版本开始，.NET 语言包括适用于 NUnit 和 xUnit 的内置模板。 对于 C++，在本示例中，选择“本机单元测试”项目，它使用 Microsoft 本机

单元测试框架。 （若要使用其他 C++ 测试框架，请参阅为 C/C++ 编写单元测试）。 对于 Python，请参阅在 Python 代码中设置单元测试以设置测试项目。

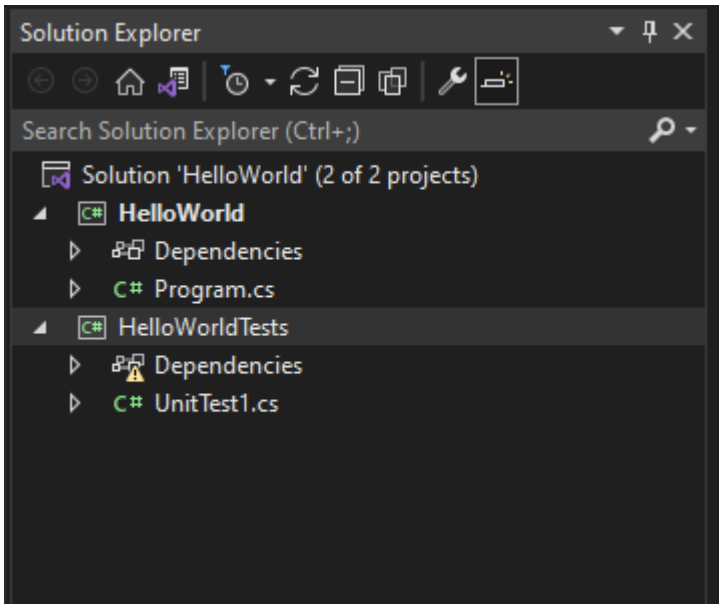
> **提示**
>
> 仅对 C# 而言，可以使用更快的方法基于代码创建单元测试项目。 有关详细信息，请参阅**创建单元测试项目和测试方法**。 若要将此方法与 .NET Core 或 .NET Standard 一起使用，需要 Visual Studio 2019 或更高版本。

下图显示了 .NET 中支持的 MSTest 单元测试。



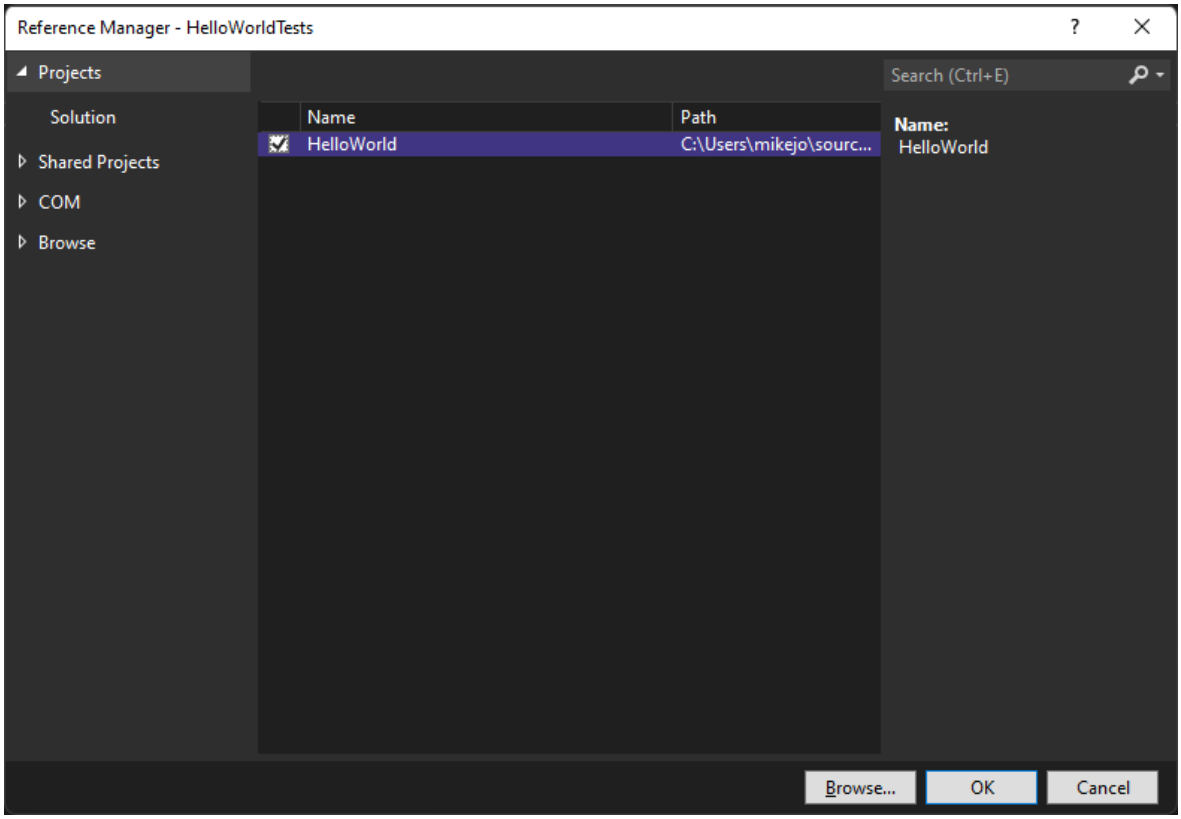
单击“下一步”，选择测试项目的名称，然后单击“创建”。

项目将添加到解决方案中。



4. 在单元测试项目中，右键单击“引用”或“依赖项”，然后选择“添加引用”或“添加项目引用”，添加对要测试的项目的引用 。

5. 选择包含待测试代码的项目，单击“确定”。



6. 向单元测试方法添加代码。

   例如，你可以通过选择与测试框架匹配的正确文档选项卡来使用以下代码：MSTest、NUnit 或 xUnit（仅在 .NET 上受支持）或 C++ Microsoft 本机单元测试框架。

MSTest

C#

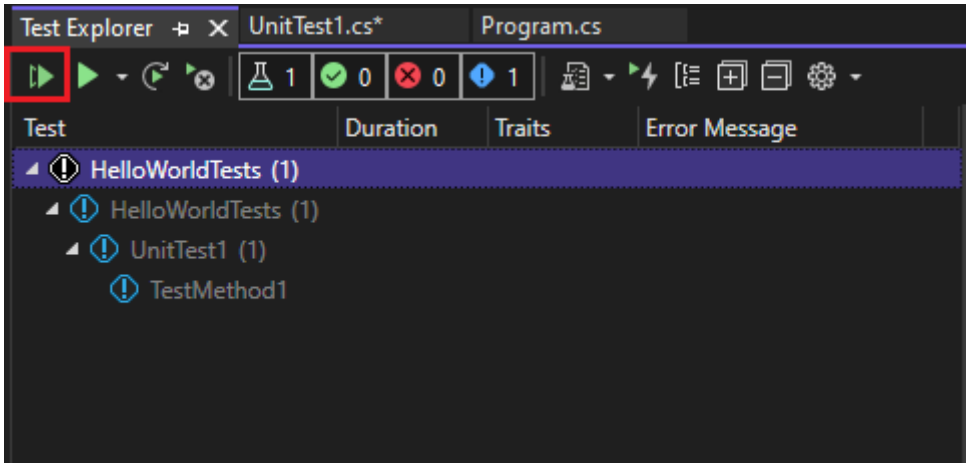
```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.IO;
using System;

namespace HelloWorldTests
{
   [TestClass]
   public class UnitTest1
   {
      private const string Expected = "Hello World!";
      [TestMethod]
      public void TestMethod1()
      {
         using (var sw = new StringWriter())
         {
            Console.SetOut(sw);
            HelloWorld.Program.Main();

            var result = sw.ToString().Trim();
            Assert.AreEqual(Expected, result);
         }
      }
   }
}
```

运行单元测试

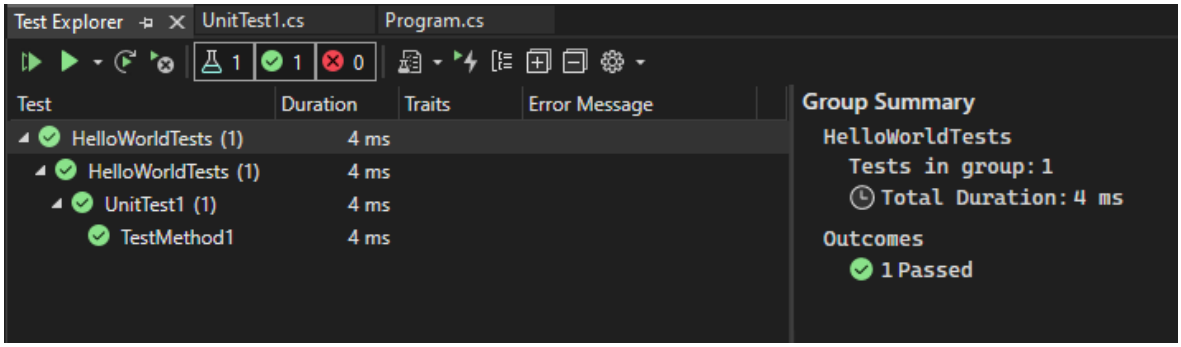
1. 打开“测试资源管理器”。

   若要打开测试资源管理器，请选择顶部菜单栏中的“测试”>“测试资源管理器”（或按 Ctrl + E，T）。

2. 单击“全部运行”（或按 Ctrl + R，V），运行单元测试。



测试完成后，绿色复选标记表示测试通过。 红色“x”图标表示测试失败。



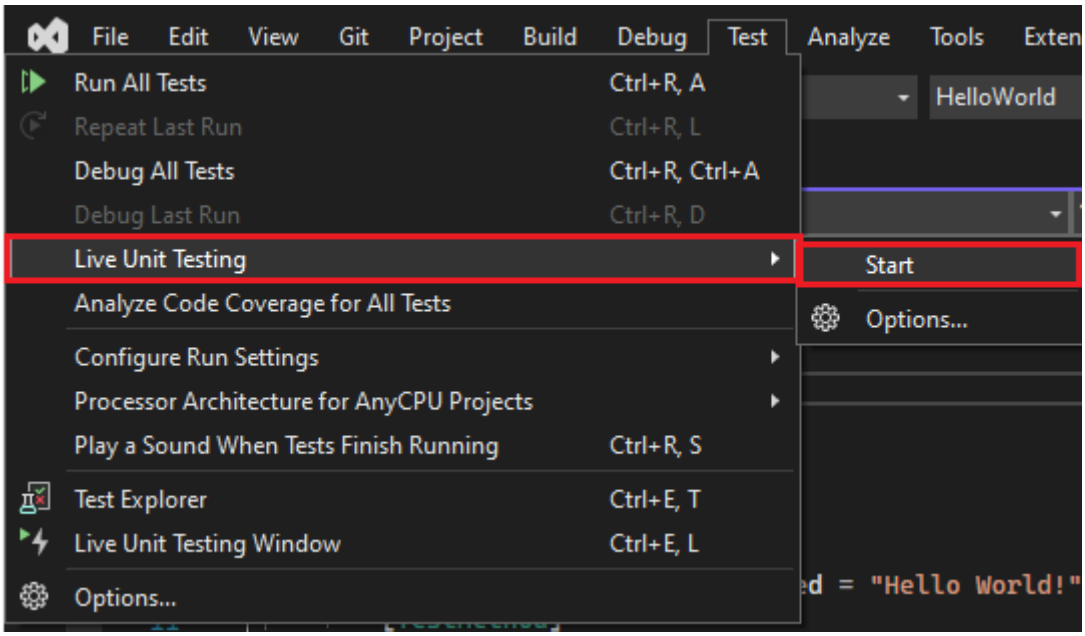
> **提示**
>
> 可以使用**测试资源管理器**从内置测试框架 (MSTest) 或第三方测试框架运行单元测试。 可以将测试分组为不同类别、筛选测试列表，以及创建、保存和运行测试播放列表。 你还可以调试测试并分析测试性能和代码覆盖率。

查看实时单元测试结果 (Visual Studio Enterprise)

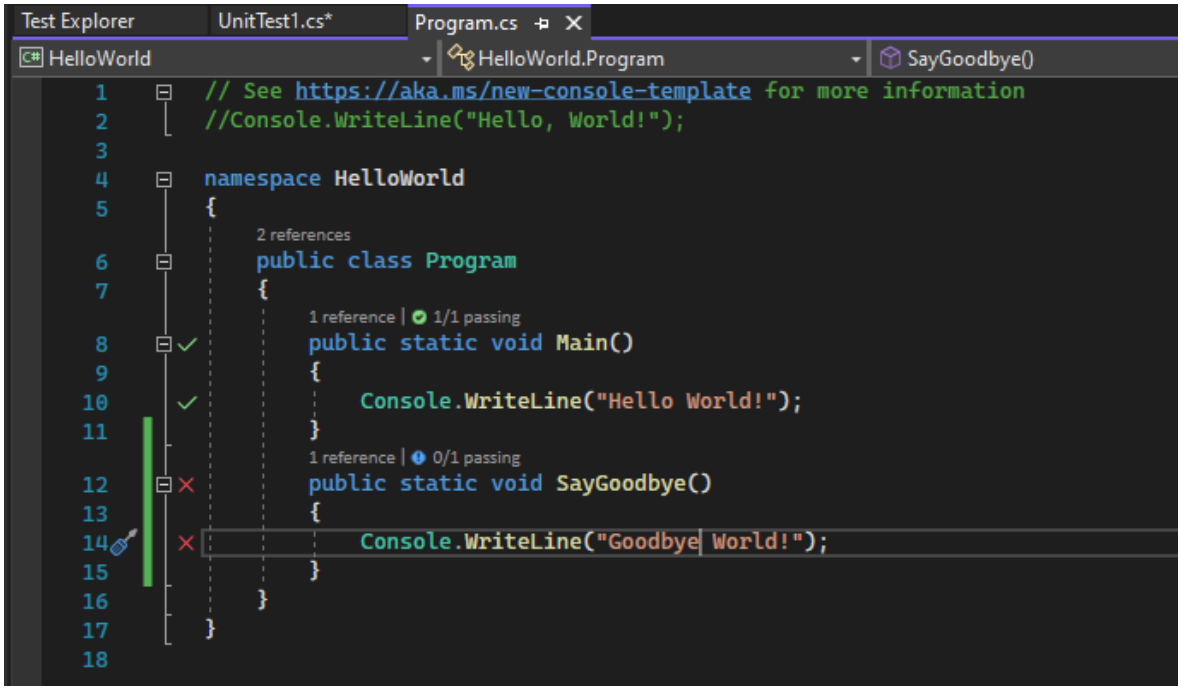
如果在 Visual Studio 2017 或更高版本中使用 MSTest、xUnit 或 NUnit 测试框架，可查看单元测试的实时结果。

> **备注**
>
> 要执行这些步骤，需要 Visual Studio Enterprise，以及 .NET 代码和以下测试框架之一：MSTest、xUnit 或 NUnit。

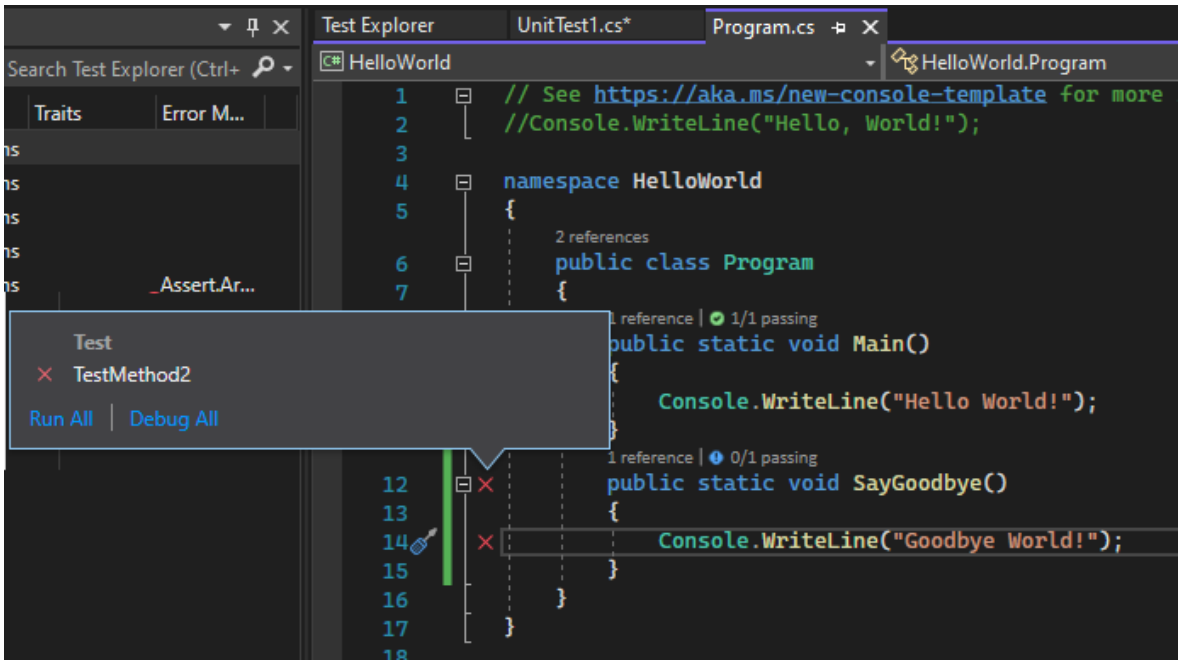
1. 选择“测试”>“Live Unit Testing”>“启动”，从“测试”菜单启用 Live Unit Testing。



2. 编写和编辑代码时，请在代码编辑器窗口中查看测试的结果。



3. 单击测试结果指示器查看详细信息，例如涵盖该方法的测试的名称。



有关 Live Unit Testing 的详细信息，请参阅 Live Unit Testing。

使用第三方测试框架

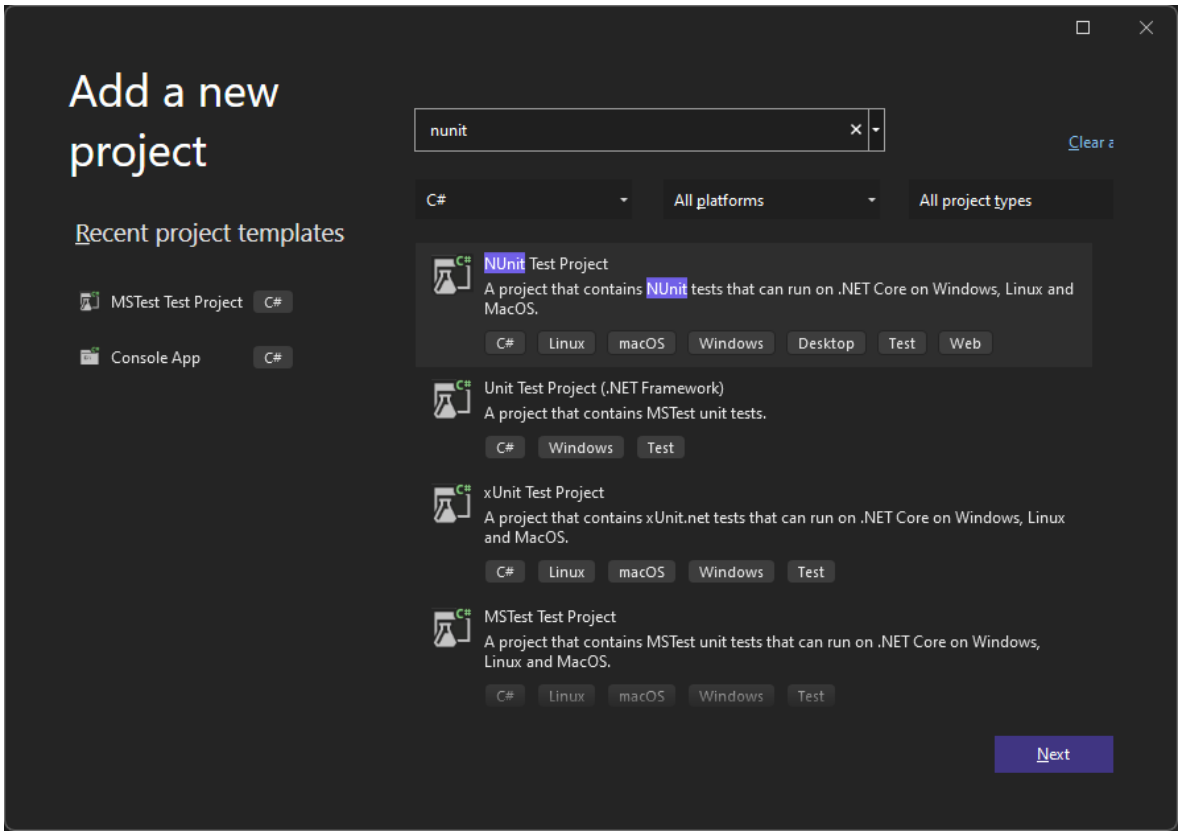
通过使用第三方测试框架（例如 NUnit、Boost 或 Google C++ 测试框架，具体取决于你的编程语言），可以在 Visual Studio 中运行单元测试。 使用第三方框架：

- 使用 NuGet 包管理器为所选框架安装 NuGet 包 。
- (.NET) 从 Visual Studio 2017 14.6 版本开始，Visual Studio 包括适用于 NUnit 和 xUnit 测试框架的预配置测试项目模板。 这些模板还包括必要的 NuGet 包以实现支持。
- (C++) 在 Visual Studio 2017 及更高版本中，已经包含了一些框架，如 Google C++ 测试框架。 有关详细信息，请参阅在 Visual Studio 中编写适用于 C/C++ 的单元测试。

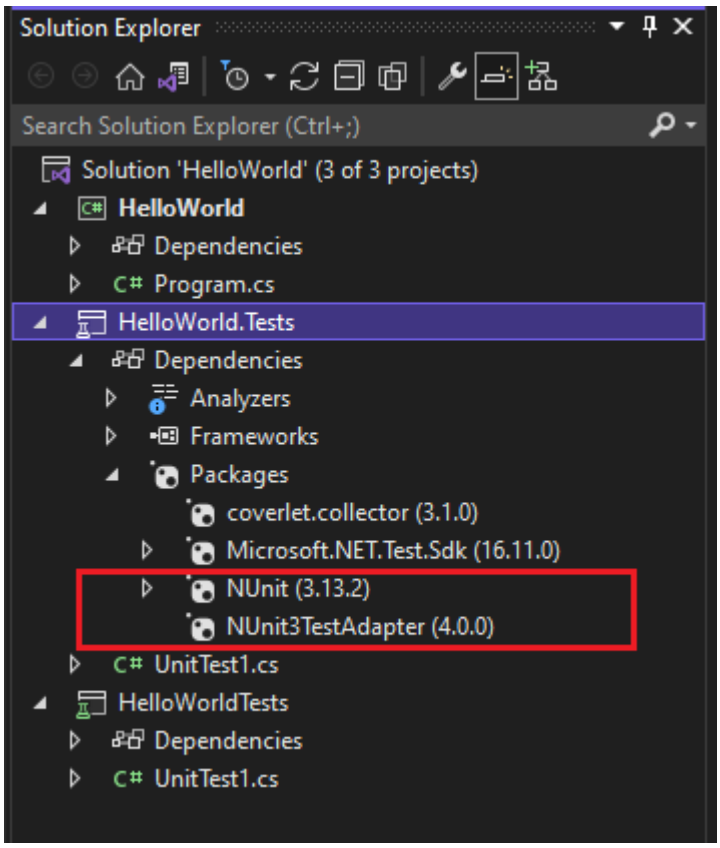
添加单元测试项目：

1. 打开包含待测试代码的解决方案。
2. 右键单击“解决方案资源管理器”中的解决方案，然后选择“添加”>“新建项目”。
3. 选择单元测试项目模板。

   在本例中，选择 NUnit



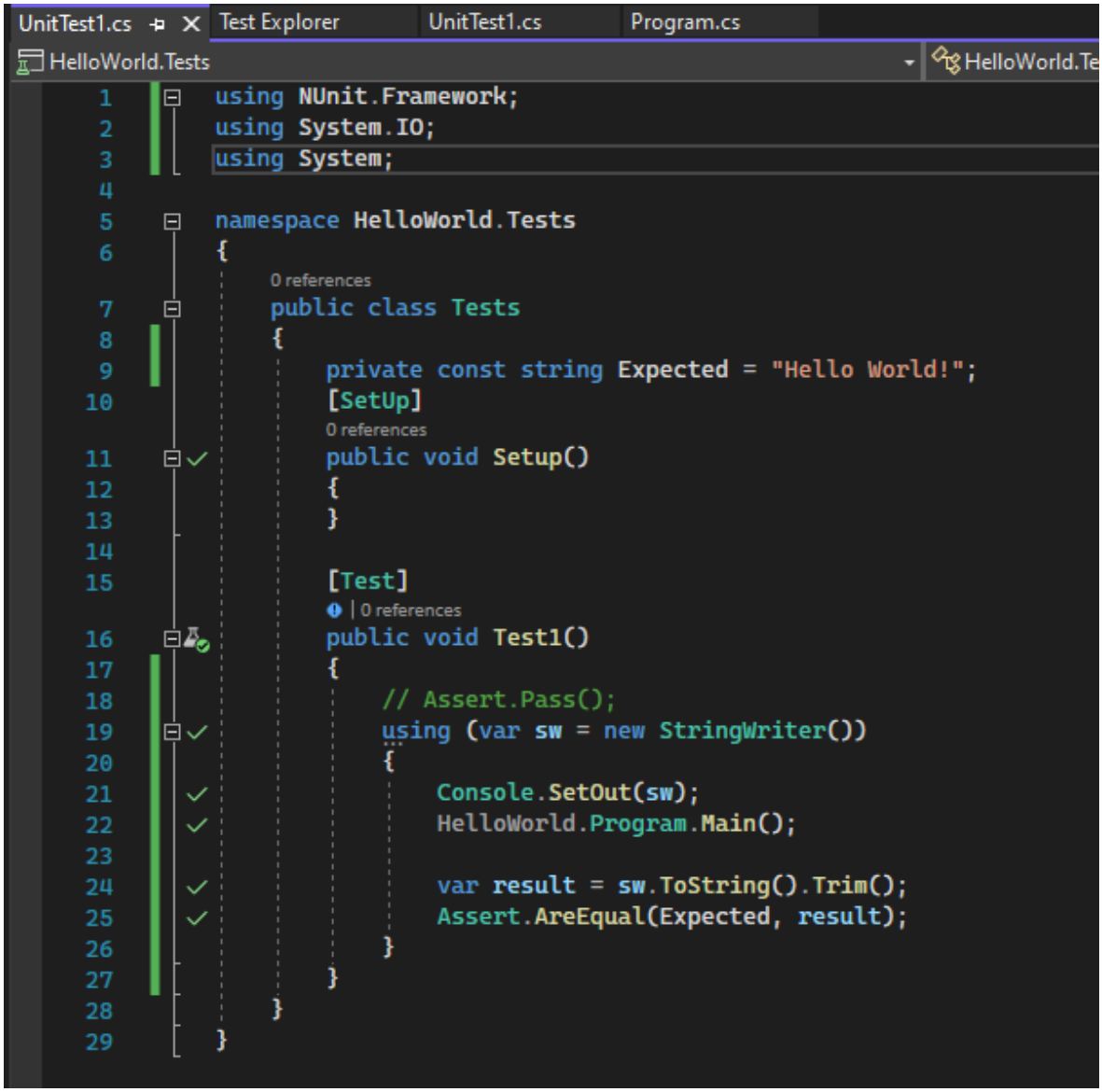
项目模板包括对 NUnit 和 NUnit3TestAdapter 的 NuGet 引用。



4. 将测试项目中的引用添加到包含待测试代码的项目中。

   右键单击“解决方案资源管理器”中的项目，然后选择“添加”>“引用”。 （还可以从“引用”或“依赖项”节点右键单击菜单来添加一个引用。）

5. 将代码添加到测试方法。



6. 从测试资源管理器运行测试，或右键单击测试代码并选择“运行测试”（或 Ctrl + R，T）。

后续步骤

单元测试基础知识

创建并运行托管代码的单元测试

编写适用于 C/C++ 的单元测试

Visual Studio 部署文档

了解如何使用 Visual Studio 部署应用程序、服务和组件

关于部署

概述

初探部署

发布概述

ASP.NET Web 应用

教程

发布工具

从 IIS 获取发布设置

通过 Git 发布

C++ 应用程序

操作指南

安装程序包

安装项目

.NET/.NET Core 应用

教程

发布工具

ClickOnce

安装程序包

操作指南

为 Windows Store 打包桌面应用

.NET Framework 桌面应用

教程

ClickOnce

安装程序包

部署到 Azure

教程

ASP.NET Core 应用到应用服务

ASP.NET 到应用服务 (.NET Framework)

ASP.NET Web 应用到 Azure Web 应用

ASP.NET Core Web 应用到 Azure 容器应用

Azure Functions

使用 GitHub 操作进行部署

使用 GitHub Actions 部署 Entity Framework Core 应用

部署 Docker 容器

操作指南

部署到 Azure 容器注册表

部署到 Azure App Service

部署到 Docker Hub

不使用 Dockerfile 进行部署

其他部署类型

操作指南

Linux 应用服务上的 Node.js 应用

容器化应用

发布 NuGet 包

使用 Visual Studio 在 .NET Framework 应用程序中创建数据库并添加表

项目 • 2024/02/01

> **备注**
>
> 数据集和相关类是 2000 年代初的旧 .NET Framework 技术，使应用程序能够在应用程序与数据库断开连接时处理内存中的数据。 它们对于使用户能够修改数据并持续更改回数据库的应用程序特别有用。 虽然数据集已被证明是一项非常成功的技术，但我们建议新的 .NET 应用程序使用 **Entity Framework Core**。 实体框架提供了一种更自然的方式来将表格数据作为对象模型，并且具有更简单的编程接口。

可以使用 Visual Studio 创建和更新 SQL Server Express LocalDB 中的本地数据库文件。 还可以通过在 Visual Studio 的 SQL Server 对象资源管理器工具窗口中执行 Transact-SQL 语句来创建数据库。 在本主题中，你将使用表设计器创建 `.mdf` 文件并添加表和键。

先决条件

要完成本演练，你需要在 Visual Studio 中安装“.NET 桌面开发”以及“数据存储和处理”工作负载。 若要安装这些程序，请打开 Visual Studio 安装程序，然后在要修改的 Visual Studio 版本旁选择“修改”（或“更多”>“修改”）。 请参阅修改 Visual Studio。

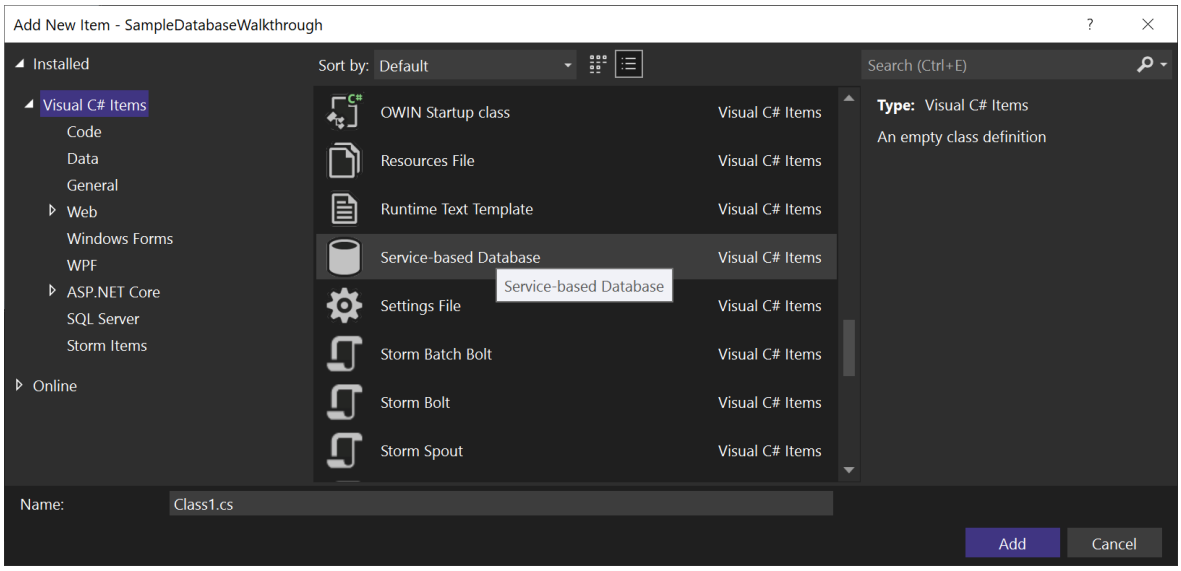
> **备注**
>
> 本文中的步骤仅适用于 .NET Framework Windows 窗体项目，不适用于 .NET Core Windows 窗体项目。

创建一个项目及本地数据库文件

1. 创建一个新的 Windows 窗体应用 (.NET Framework) 项目，并将其命名为 SampleDatabaseWalkthrough。

2. 在菜单栏上，依次选择“项目”>“添加新项”。 如果显示带有文件名框的小对话框，请选择“显示所有模板”。

3. 在项模板列表中，向下滚动并选择“基于服务的数据库”。

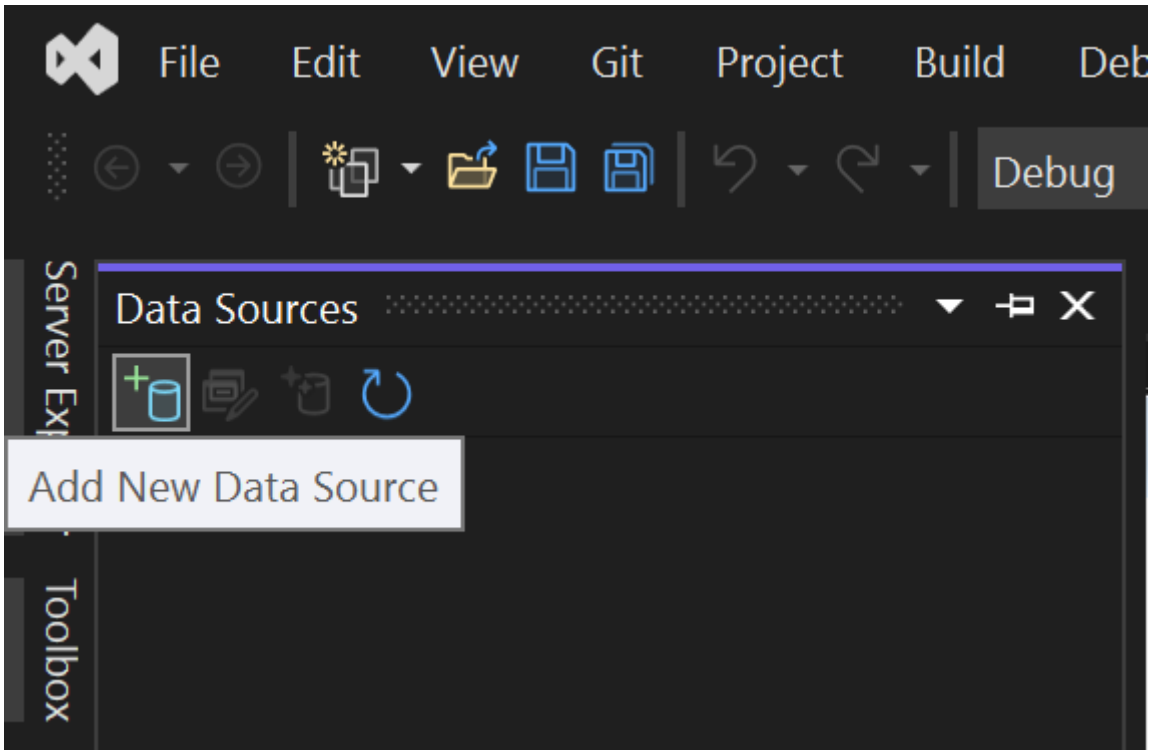


4. 将数据库命名为 SampleDatabase.mdf，然后单击“添加”。

添加数据源

1. 如果“数据源”窗口未打开，通过按 Shift + Alt + D 或选择菜单栏上的“查看”>“其他窗口”“数据源”>来将其打开。

2. 在“数据源”窗口中，选择“添加新的数据源”。



“数据源配置”向导随即打开。

3. 在“选择数据源类型”页上，选择“数据库”，然后单击“下一步”。

4. 在“选择数据库模型”页上，选择“下一步”以接受默认（数据集）。

5. 在“选择数据连接”页上，在下拉列表中选择 SampleDatabase.mdf 文件，然后选择“下一步”。

6. 在“将连接字符串保存到应用程序配置文件”页上，选择“下一步”。

7. 在“选择数据库对象”页上，你将看到提示数据库不包含任何对象的消息。 选择“完成”。

查看数据连接的属性

可以通过打开数据连接的“属性”窗口来查看 SampleDatabase.mdf 文件的一些属性：

- 选择“查看”>“SQL Server 对象资源管理器”（或 **Ctrl**+\，**Ctrl**+**S**）以打开“SQL Server 对象资源管理器”窗口。 展开“(localdb)\MSSQLLocalDB”>“数据库”，然后右键单击 SampleDatabase.mdf（可能是以完整路径列出）并选择“属性”。

- 或者，如果窗口尚未打开，则可以选择“查看”>“服务器资源管理器”。 通过展开“数据连接”节点、右键单击 SampleDatabase.mdf，然后选择“属性”可以打开“属性”窗口。

  > **提示**
  >
  > 如果无法展开“数据连接”接口或未列出 SampleDatabase.mdf 连接，请选择“服务器资源管理器”工具栏中的“连接到数据库”按钮。 在“添加连接”对话框中，确保在“数据源”下选择了“Microsoft SQL Server 数据库文件”，然后浏览到 SampleDatabase.mdf 文件并选择该文件。 通过选择“确定”完成添加连接。

若要查看连接字符串，可以在解决方案资源管理器中打开 *App.config* 文件。 应会在 `connectionStrings` 元素下看到类似于以下代码的条目：

XML

```
    <connectionStrings>
        <add
name="SampleDatabaseWalkthrough.Properties.Settings.SampleDatabaseConnection
String"
            connectionString="Data Source=
(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\SampleDatabase.mdf;I
ntegrated Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
```

使用表设计器创建表和键

在本节中，你将创建两个表，每个表中有一个主键和几行示例数据。 你还将创建外键以指定一个表中的记录如何对应于另一个表中的记录。

创建 Customers 表

1. 在“服务器资源管理器”或“SQL Server 对象浏览器”中，依次展开“数据连接”节点和“SampleDatabase.mdf”节点。

2. 右键单击“表”，然后选择“添加新表” 。

   “表设计器”将打开并显示一个网格，其中有一个默认行，表示所创建表中的一列。 通过向网格中添加行，即可在表中添加列。

3. 在网格中，为下列各个条目添加行：

   展开表

列名称	数据类型	允许空
`CustomerID`	`nchar(5)`	False（清除）
`CompanyName`	`nvarchar(50)`	False（清除）
`ContactName`	`nvarchar (50)`	True（已选定）
`Phone`	`nvarchar (24)`	True（已选定）

4. 在 `CustomerID` 行上右键单击，然后选择“设置主键”。

5. 在默认行 (`Id`) 上右键单击，然后选择“删除”。

6. 通过更新脚本窗格的第一行来命名 Customers 表，与以下示例相匹配：

   ```
   SQL
   ```

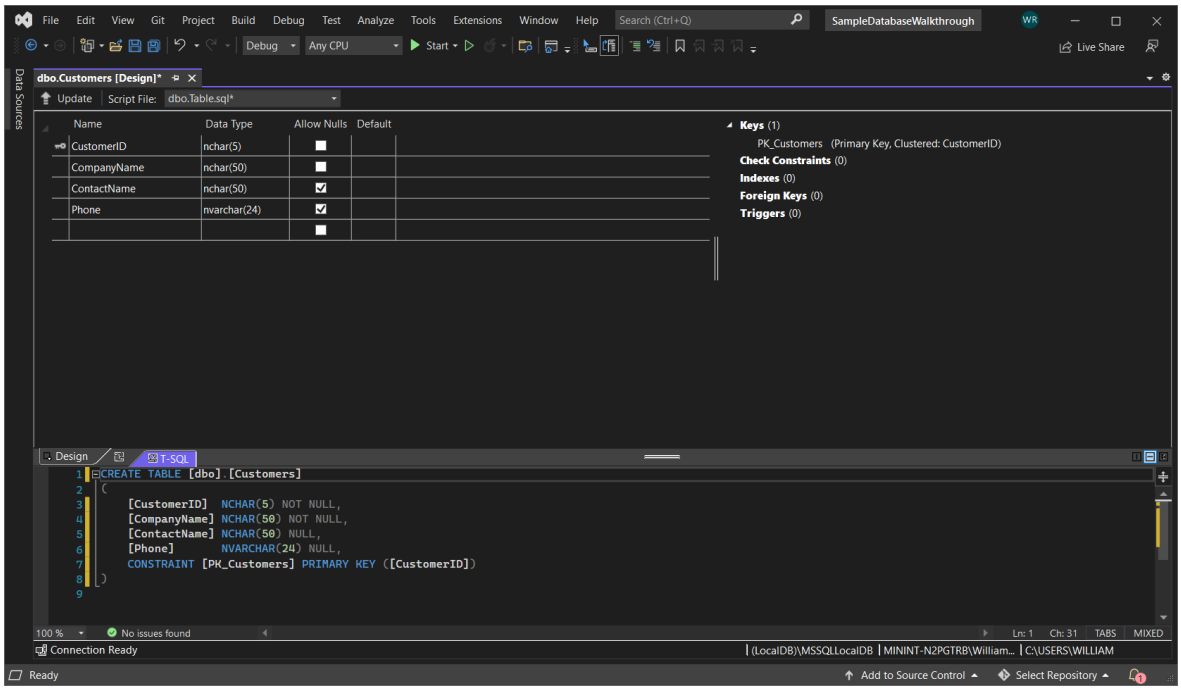
   ```
   CREATE TABLE [dbo].[Customers]
   ```

7. 向“客户”表添加索引约束。 在 `Phone` 行末尾添加一个逗号，然后在右括号前添加以下示例：

   ```
   SQL
   ```

   ```
   CONSTRAINT [PK_Customers] PRIMARY KEY ([CustomerID])
   ```

   你应看到与下面类似的内容：



8. 在“表设计器”的左上角中，选择“更新”，或按 Shift + Alt + U。

9. 在“预览数据库更新”对话框中，选择“更新数据库”。

   在本地数据库文件中创建了 Customers 表。

创建 Orders 表

1. 添加另一个表，然后在下表中为每个条目添加行：

展开表

列名称	数据类型	允许空
`OrderID`	`int`	False（清除）
`CustomerID`	`nchar(5)`	False（清除）
`OrderDate`	`datetime`	True（已选定）
`OrderQuantity`	`int`	True（已选定）

2. 将“OrderID”设置为主键，然后删除默认行。

3. 通过更新脚本窗格的第一行来命名 Orders 表，与以下示例相匹配：

```SQL
CREATE TABLE [dbo].[Orders]
```

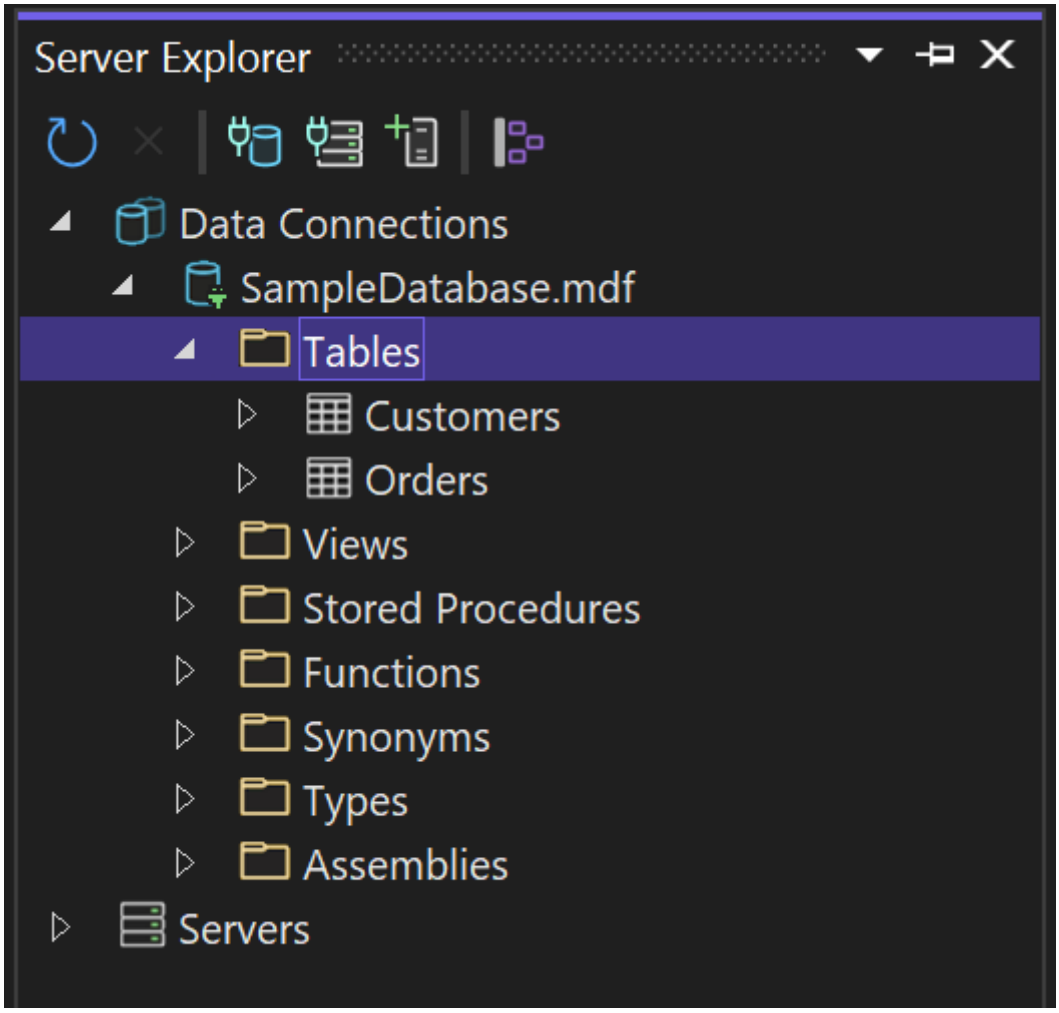
4. 向“客户”表添加索引约束。 在 `OrderQuantity` 行末尾添加一个逗号，然后在右括号前添加以下示例：

   ```
   SQL

   CONSTRAINT [PK_Orders] PRIMARY KEY ([OrderId])
   ```

5. 在“表设计器”的左上角中，选择“更新”，或按 Shift + Alt + U。

6. 在“预览数据库更新”对话框中，选择“更新数据库”。

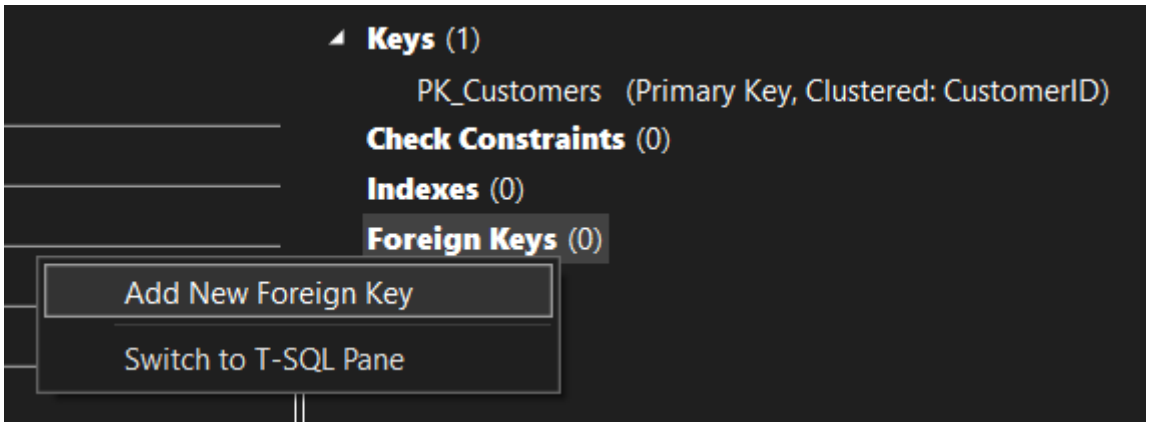
   在本地数据库文件中创建了 Orders 表。 如果在服务器资源管理器中展开“表”节点，则可以看到两个表：

   如果看不到，则点击“刷新”工具栏按钮。

创建外键

1. 在 Orders 表的“表设计器”网格的右侧上下文窗格中，右键单击“外键”，然后选择“添加新的外键”。



2. 在显示的文本框中，将文本“ToTable”替换为“Customers” 。

3. 在 T-SQL 窗格中，更新最后一行以与以下示例相匹配：

```
SQL

CONSTRAINT [FK_Orders_Customers] FOREIGN KEY ([CustomerID]) REFERENCES
[Customers]([CustomerID])
```

4. 在“表设计器”的左上角中，选择“更新”(Shift + Alt + U)。

5. 在“预览数据库更新”对话框中，选择“更新数据库”。

   将创建外键。

将数据填入表中

1. 在“服务器资源管理器”或“SQL Server 对象资源管理器”中，展开示例数据库的节点。

2. 打开“表”节点的快捷菜单，选择“刷新”，然后展开“表”节点。

3. 打开“客户”表的快捷菜单，然后选择“显示表数据”或“查看数据”。

4. 为一些客户添加任何所需数据。

   你可以指定任意五个字符作为客户 ID，但至少选择一个能记住的以便稍后在此过程中使用。

5. 打开“订单”表的快捷菜单，然后选择“显示表数据”或“查看数据”。

6. 为一些订单添加数据。 输入每一行时，该行将保存在数据库中。

> ⓘ **重要**

请确保所有订单 ID 和订单数量是整数，并且每个客户 ID 与 Customers 表中的“CustomerID”列中指定的值匹配。

祝贺你！ 现在已经了解如何创建表、将其与外键链接并添加数据。

相关内容

- 在 Visual Studio 中访问数据