

在 Visual Studio 中使用 .NET Core 测试 .NET Standard 库

2019/12/24 • 👁 👁 👁

本文内容

[创建单元测试项目](#)

[添加并运行单元测试方法](#)

[处理测试失败](#)

[测试库的发行版本](#)

[请参阅](#)

在[在 Visual Studio 中生成 .NET Standard 库](#)中，创建了一个简单的类库，用于向 [String](#) 类添加扩展方法。现在，将创建一个单元测试，用于确保此类库能够按预期运行。向在上一篇文章中创建的解决方案添加单元测试项目。

创建单元测试项目

若要创建单元测试项目，请执行以下操作：

1. 打开在[在 Visual Studio 中生成 .NET Standard 库](#)一文中创建的 ClassLibraryProjects 解决方案。
2. 将名为“StringLibraryTest”的新单元测试项目添加到解决方案。
 - a. 在“解决方案资源管理器”中右键单击解决方案并选择“添加” > “新建项目”。
 - b. 在“添加新项目”页面，在搜索框中输入“mstest”。从“语言”列表中选择“C#”或“Visual Basic”，然后从“平台”列表中选择“所有平台”。选择“MsTest 测试项目 (.NET Core)”模板，然后选择“下一步”。
 - c. 在“配置新项目”页面，在“项目名称”框中输入“StringLibraryTest”。然后选择“创建”。

📌 备注

除了 MSTest 之外，还可以在 Visual Studio 中为 .NET Core 创建 xUnit 和 NUnit 测试项目。

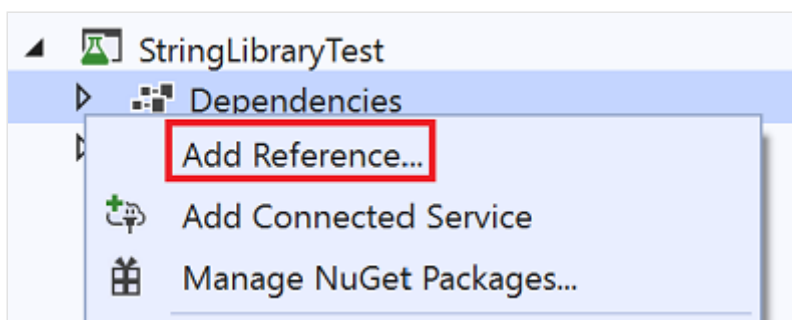
3. 此时，Visual Studio 会创建项目，并在具有以下代码的代码窗口中打开类文件：

C#	复制
<pre>using Microsoft.VisualStudio.TestTools.UnitTesting; namespace StringLibraryTest { [TestClass] public class UnitTest1 { [TestMethod] public void TestMethod1() { } } }</pre>	

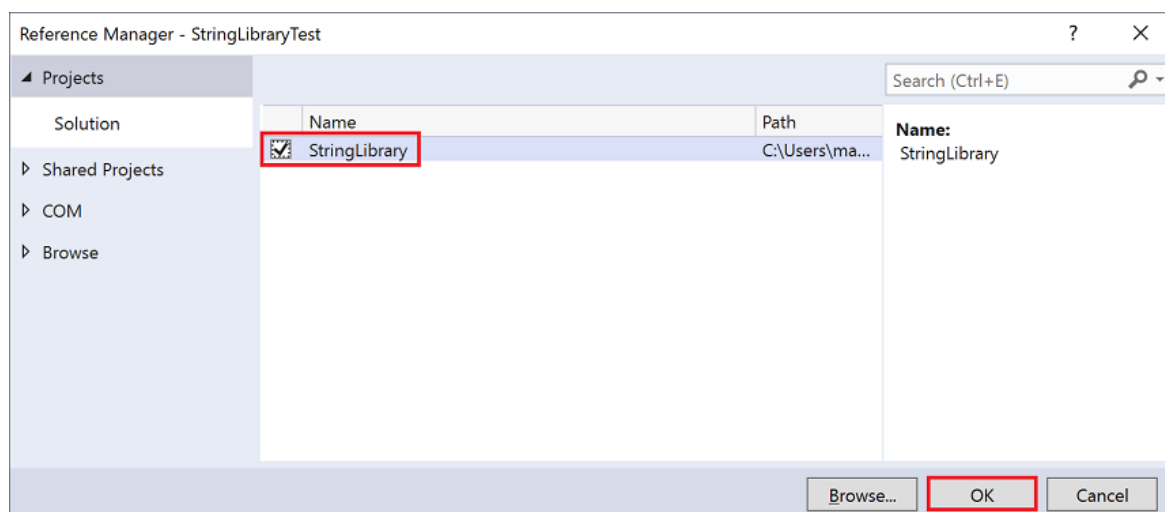
单元测试模板创建的源代码负责执行以下操作：

- 它会导入 [Microsoft.VisualStudio.TestTools.UnitTesting](#) 命名空间，其中包含用于单元测试的类型。
- 向 UnitTest1 类应用 [TestClass](#) 特性。测试类中标记有 [TestMethod](#) 属性的所有测试方法都会在单元测试运行时自动执行。
- 它应用 [TestMethod](#) 属性，将 C# 中的 TestMethod1 或将 Visual Basic 中的 TestSub 定义为在单元测试运行时自动执行的测试方法。

4. 在“解决方案资源管理器”中，右键单击“StringLibraryTest”项目的“依赖项”节点，并从上下文菜单中选择“添加引用”。



5. 在“引用管理器”对话框中，展开“项目”节点，并选中“StringLibrary”旁边的框。添加对 StringLibrary 程序集的引用后，编译器可以查找 StringLibrary 方法。选择“确定”按钮。这会添加对类库项目 StringLibrary 的引用。



添加并运行单元测试方法

运行单元测试时，Visual Studio 执行单元测试类（对其应用了 [TestClassAttribute](#) 特性的类）中标记有 [TestMethodAttribute](#) 特性的所有方法。当第一次遇到测试不通过或测试方法中的所有测试均已成功通过时，测试方法终止。

最常见的测试调用 [Assert](#) 类的成员。许多断言方法至少包含两个参数，其中一个是预期的测试结果，另一个是实际的测试结果。下表显示了 [Assert](#) 类最常调用的一些方法：

断言方法	函数
<code>Assert.AreEqual</code>	验证两个值或对象是否相等。如果值或对象不相等，则断言失败。
<code>Assert.AreSame</code>	验证两个对象变量引用的是否是同一个对象。如果这些变量引用不同的对象，则断言失败。
<code>Assert.IsFalse</code>	验证条件是否为 <code>false</code> 。如果条件为 <code>true</code> ，则断言失败。
<code>Assert.IsNotNull</code>	验证对象是否不为 <code>null</code> 。如果对象为 <code>null</code> ，则断言失败。

还可以在测试方法中使用 [ThrowsException](#) 方法来指示它应引发的异常的类型。如果未引发指定异常，则测试不通过。

测试 `StringLibrary.StartsWithUpper` 方法时，需要提供许多以大写字母开头的字符串。在这种情况下，此方法应返回 `true`，以便可以调用 [IsTrue](#) 方法。同样，需要提供许多以非大写字母开头的字符串。在这种情况下，此方法应返回 `false`，以便可以调用 [IsFalse](#) 方法。

由于库方法处理的是字符串，因此还需要确保它能够成功处理[空字符串 \(String.Empty\)](#)（不含字符且 [Length](#) 为 0 的有效字符串）和 `null` 字符串（尚未初始化的字符串）。如果对 [String](#) 实例调用 `StartsWithUpper` 作为扩展方法，无法向其传递 `null` 字符串。不过，还可以直接将其作为静态方法进行调用，并向其传递一个 [String](#) 自变量。

将定义三个方法，每个方法都会对字符串数组中的各个元素反复调用它的 [Assert](#) 方法。由于测试方法在第一次遇到测试不通过时会立即失败，因此将调用方法重载，以便传递字符串来指明方法调用中使用的字符串值。

创建测试方法：

1. 将 UnitTest1.cs 或 UnitTest1.vb 代码窗口中的代码替换为以下代码：

C#	 复制
<pre>using System; using Microsoft.VisualStudio.TestTools.UnitTesting; using UtilityLibraries; namespace StringLibraryTest { [TestClass] public class UnitTest1 { [TestMethod] public void TestStartsWithUpper() { // Tests that we expect to return true. string[] words = { "Alphabet", "Zebra", "ABC", "Αθήνα", "Москва" }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsTrue(result, String.Format("Expected for '{0}': true; Actual: {1}", word, result)); } } [TestMethod] public void TestDoesNotStartWithUpper() { // Tests that we expect to return false. string[] words = { "alphabet", "zebra", "abc", "αυτοκινητοβιομηχανία", "государство", "1234", ".", ";", " " }; foreach (var word in words) { bool result = word.StartsWithUpper(); Assert.IsFalse(result, String.Format("Expected for '{0}': false; Actual: {1}", word, result)); } } [TestMethod]</pre>	

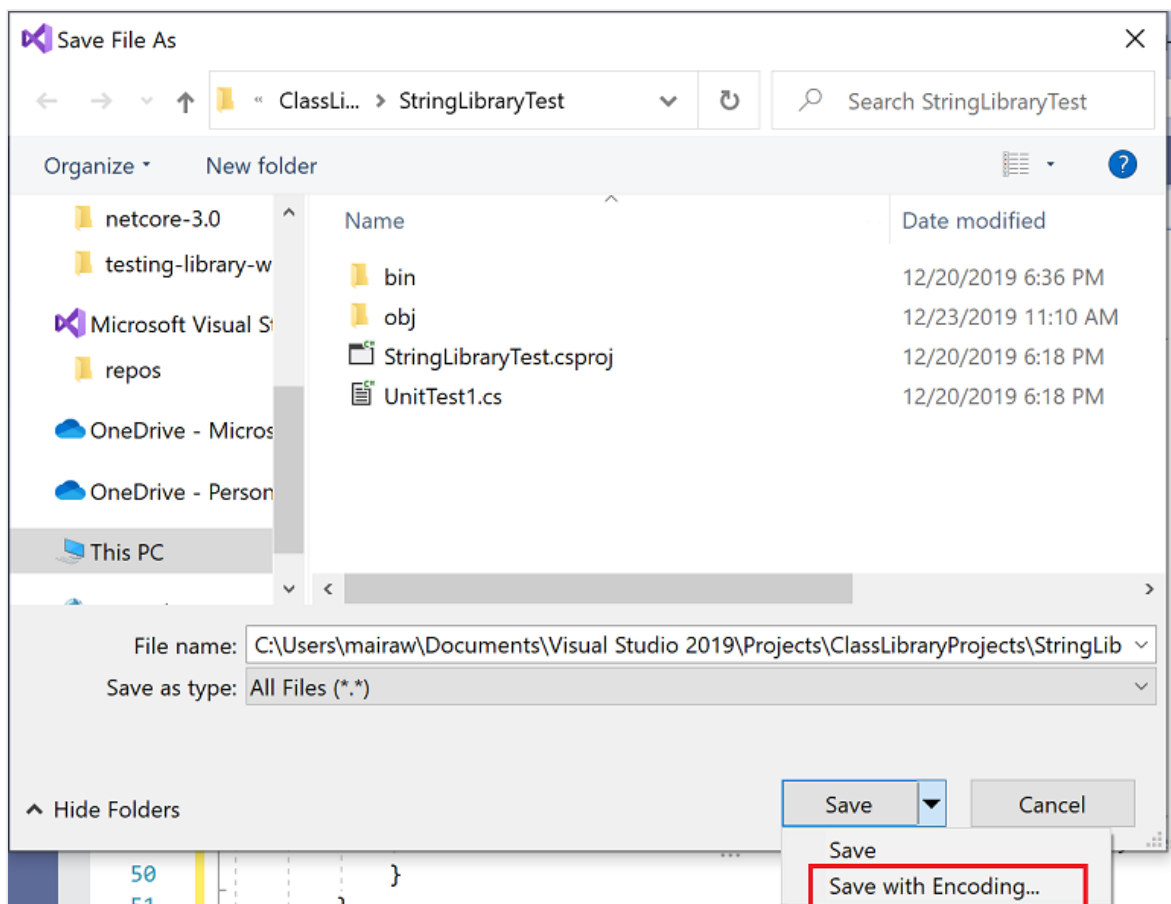
```

public void DirectCallWithNullOrEmpty()
{
    // Tests that we expect to return false.
    string[] words = { string.Empty, null };
    foreach (var word in words)
    {
        bool result = StringLibrary.StartsWithUpper(word);
        Assert.IsFalse(result,
            String.Format("Expected for '{0}': false;
Actual: {1}",
                                word == null ? "<null>" : word,
result));
    }
}
}
}

```

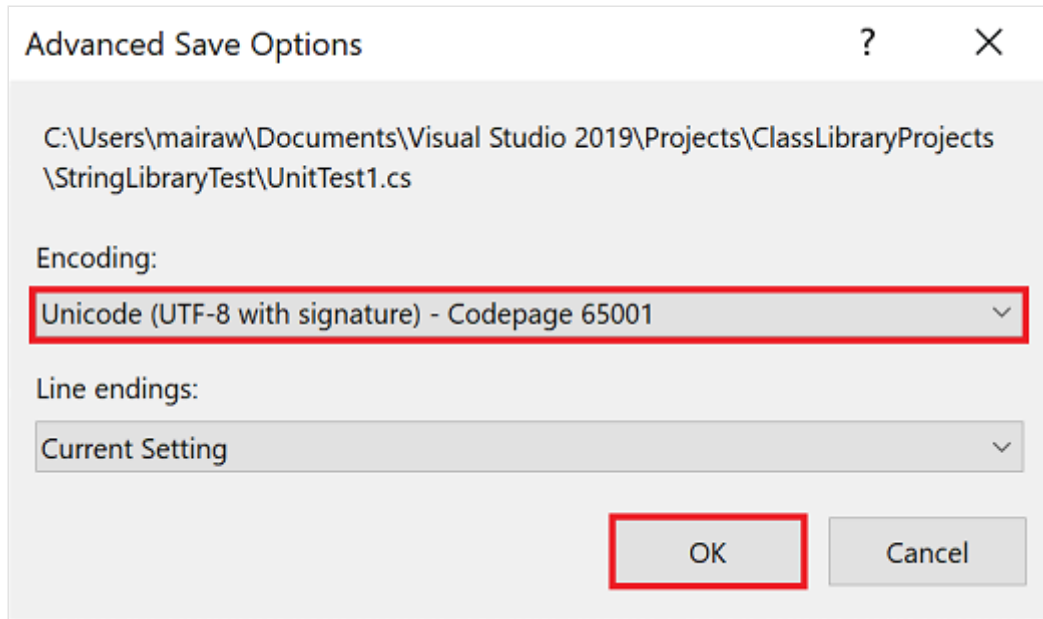
TestStartsWithUpper 方法中的大写字母的测试包括希腊文大写字母 alpha (U+0391) 和西里尔文大写字母 EM (U+041C)。 TestDoesNotStartWithUpper 方法中的小写字母的测试包括希腊文小写字母 alpha (U+03B1) 和西里尔文小写字母 Ghe (U+0433)。

2. 在菜单栏上，选择“文件” > “将 UnitTest1.cs 另存为”或“文件” > “将 UnitTest1.vb 另存为”。在“文件另存为”对话框中，选择“保存”按钮旁边的箭头，然后选择“保存时使用编码”。



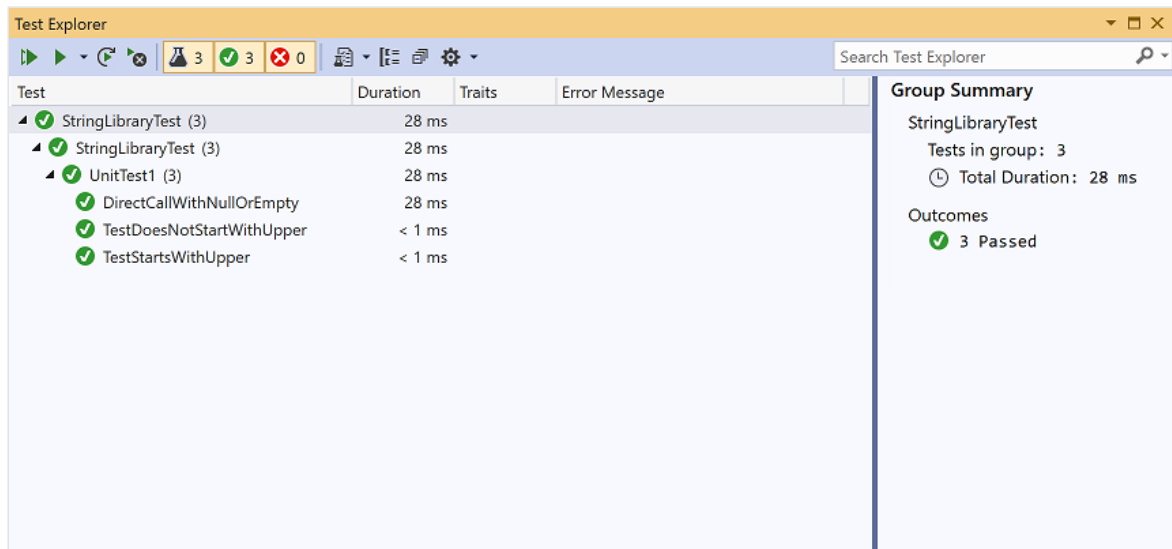
3. 在“确认另存为”对话框中，选择“是”按钮，保存文件。

- 在“高级保存选项”对话框的“编码”下拉列表中，选择“Unicode (UTF-8 带签名) - 代码页 65001”，然后选择“确定”。



如果无法将源代码保存为 UTF8 编码文件，Visual Studio 可能会将其另存为 ASCII 文件。在这种情况下，运行时将无法准确解码 ASCII 范围以外的 UTF8 字符，且测试结果也会不正确。

- 在菜单栏上，选择“测试” > “运行” > “所有测试”。此时，“测试资源管理器”窗口打开并显示测试已成功运行。“通过的测试”部分列出了三个测试，“摘要”部分报告了测试运行结果。



处理测试失败

由于运行的测试均通过，因此需进行少量改动，以使其中一个测试方法失败：

- 通过修改 `TestDoesNotStartWithUpper` 方法中的 `words` 数组来包含字符串“Error”。
由于 Visual Studio 将在生成运行测试的解决方案时自动保存打开的文件，因此无需

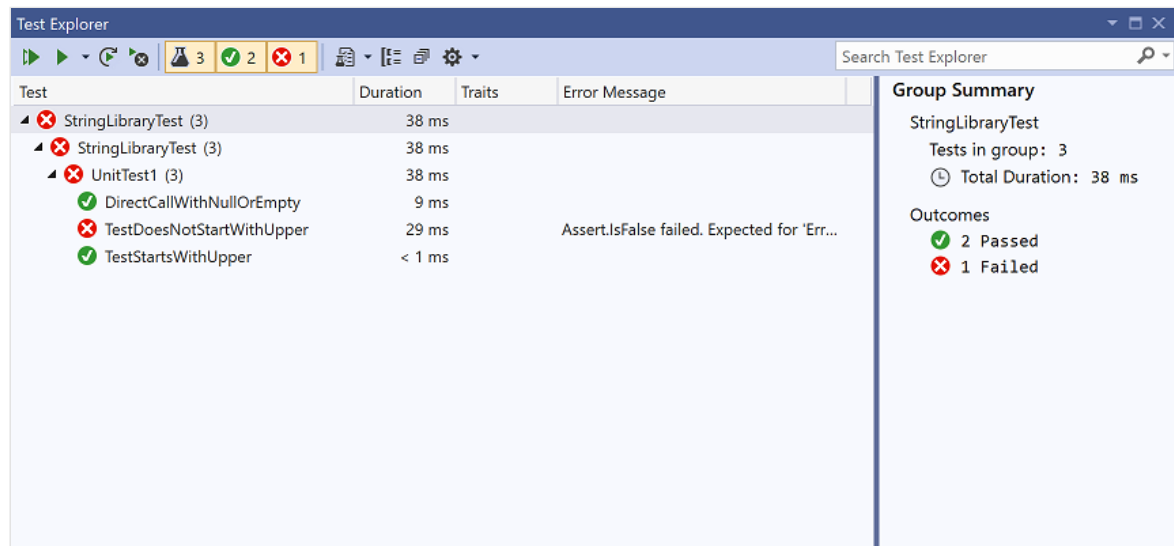
手动保存。

C#

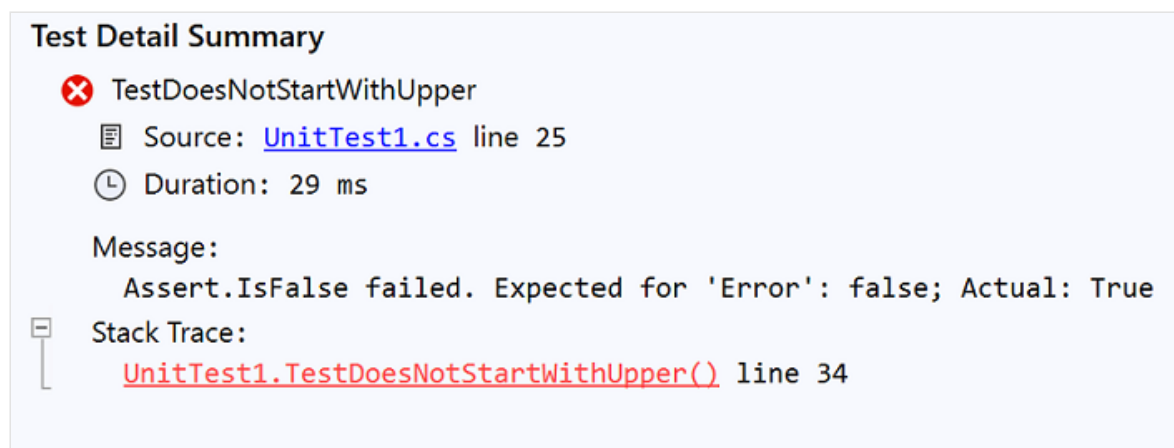
复制

```
string[] words = { "alphabet", "Error", "zebra", "abc",  
"αυτοκινητοβιομηχανία", "государство",  
"1234", ".", ";", " " };
```

2. 从菜单栏中选择“测试” > “运行” > “所有测试”，运行测试。“测试资源管理器”窗口指示有两个测试成功，还有一个失败。



3. 选择失败的测试，TestDoesNotStartWith。 “测试资源管理器”窗口显示断言生成的消息：“Assert.IsFalse 失败。“Error”应返回 false；实际返回 True”。由于此次失败，数组中“Error”之后的所有字符串都未进行测试。



4. 撤消在步骤 1 中执行的修改并删除字符串“Error”。 重新运行测试，测试将通过。

测试库的发行版本

现已测试库的调试版本。至此，测试已全部通过，且已充分测试库，应对库的发布版本再运行一次这些测试。许多因素（包括编译器优化）有时可能会导致调试版本和发行版

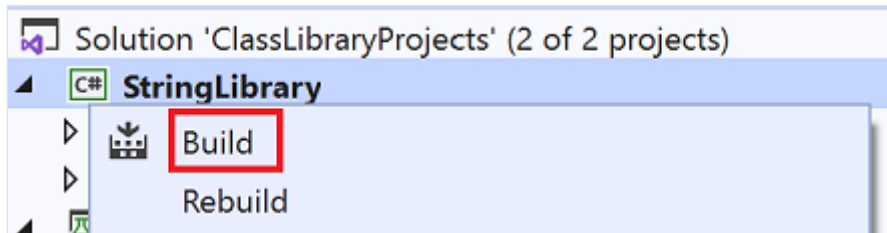
本出现行为差异。

若要测试发行版本，请执行以下操作：

1. 在 Visual Studio 工具栏中，将生成配置从“**调试**”更改为“**发行**”。



2. 在“解决方案资源管理器”中，右键单击“StringLibrary”项目，从上下文菜单中选择“生成”，重新编译库。




3. 从菜单栏中选择“测试” > “运行” > “所有测试”，运行单元测试。测试通过。

至此，已完成对库的测试，下一步就是使其可供调用方使用。可以将类库与一个或多个应用程序捆绑在一起，也可以 NuGet 包的形式分发类库。有关详细信息，请参阅[使用 .NET Standard 类库](#)。

请参阅

- [单元测试基础知识 - Visual Studio](#)

此页面有帮助吗？

 是  否
