

Visual Studio 2005 Visual C#

Copyright© 2016 Microsoft Corporation

本文档中的内容已停用，将不再更新且不支持。某些链接可能无效。已停用的内容表示此内容的最近更新版本。

Visual Studio

Visual Studio 是一套完整的工具，用于生成桌面和基于团队的企业级 Web 应用程序。除了生成高性能的桌面应用程序外，还可以使用 Visual Studio 基于组件的强大开发工具和其他技术，简化基于团队的企业级解决方案的设计、开发和部署。

本节内容

[Visual Studio 简介](#)

找到有关 Visual Studio 新增功能的更多信息，了解有关 .NET Framework 的更多信息，并查找指向此版本 Visual Studio 入门的指针。

[Visual Studio 集成开发环境](#)

找到有关设计、开发、调试、测试、部署和管理用 Visual Studio 创建的应用程序的信息。

[基于 Windows 的应用程序、组件和服务](#)

确定生成应用程序和组件时使用的工具和技术，以及可使用 Visual Studio 创建的项。

[Visual Studio 中的 .NET Framework 编程](#)

了解在 Visual Basic、Visual C# 和 Visual J# 中开发应用程序时如何使用 .NET Framework。

[Visual Basic](#)

了解 Visual Basic 的新增功能，并研究如何使用 Visual Basic 开发应用程序。

[Visual C#](#)

了解 Visual C# 的新增功能，并研究如何使用 Visual C# 开发应用程序。

[Visual C++](#)

找到有关 Visual C++ 的新增功能的信息，以及发现如何使用 Visual C++ 开发应用程序。

[Visual J#](#)

找到有关 Visual J# 的信息，Visual J# 是一种工具，供 Java 语言程序员用于生成在 .NET Framework 上运行的应用程序和服务。

[JScript](#)

找到有关 JScript .NET(真正面向对象的脚本语言)的信息。

[Visual Web Developer](#)

了解 Visual Web Developer 并研究如何使用 Visual Web Developer 创建 Web 应用程序。

[Visual Studio Team System](#)

了解 Visual Studio 2005 Team System，这是一个高效、集成且可扩展的软件开发生命周期工具平台，可以帮助软件团队在整个软件开发过程中提高沟通和协作能力。

[Visual Studio Tools for Office](#)

了解如何创建商业应用程序，以利用 Microsoft Office 2003 的强大功能对信息进行收集、分析、操作或呈现。

[智能设备开发](#)

了解如何开发在基于 Windows CE 的智能设备(如 Pocket PC 和 Smartphone)上运行的软件。

[工具和功能](#)

了解 Crystal Reports、Windows Server 功能编程以及应用程序验证工具。

[.NET Framework 示例](#)

定位此版本的 Visual Studio 中提供的最新示例应用程序和示例。

[快速入门](#)

了解 .NET Framework SDK 附带的教程。

[.NET Framework 词汇表](#)

了解 .NET Framework 中使用的常用术语的定义。

相关章节

[Visual Studio SDK](#)

找到有关 Visual Studio 软件开发工具包 (SDK)(提供扩展性选项和工具包)的信息。

Visual C#

Microsoft Visual C# 2005(读作 C sharp)是一种编程语言，它是为生成在 .NET Framework 上运行的多种应用程序而设计的。C# 简单、功能强大、类型安全，而且是面向对象的。C# 凭借它的许多创新，在保持 C 样式语言的表示形式和优美的同时，实现了应用程序的快速开发。

Visual Studio 支持 Visual C#，这是通过功能齐全的代码编辑器、项目模板、设计器、代码向导、功能强大且易于使用的调试器以及其他工具实现的。通过 .NET Framework 类库，可以访问多种操作系统服务和其他有用的精心设计的类，这些类可显著加快开发周期。

本节内容

[Visual C# 入门](#)

针对初次接触此语言或初次接触 Visual Studio 的程序员，介绍 C# 2.0 的功能，并提供查找关于 Visual Studio 的帮助的路标。这也是“如何实现”页面所在的位置。

[使用 Visual C# IDE](#)

介绍 Visual C# 开发环境。

[用 Visual C# 编写应用程序](#)

通过指向更详细文档的链接，为使用 C# 和 .NET Framework 的常见编程任务提供高级指导。

[迁移到 Visual C#](#)

将 C# 语言与 Java 和 C++ 进行对比，并介绍如何使用 Java Language Conversion Assistant 将 Java 和 Visual J++ 应用程序转换为 Visual C#。

[C# 编程指南](#)

提供有关如何使用 C# 语言结构的信息和实例。

[C# 参考](#)

提供关于以下内容的详细参考信息：C# 编程概念、关键字、类型、运算符、属性、处理器指令、编译器开关，以及编译器错误和警告。

[C# 语言规范](#)

指向 C# 规范的最新版本(Microsoft Word 格式)的链接。

[Visual C# 示例](#)

演示如何使用 Visual C# 编程的源代码示例。

相关章节

[C# 2.0 语言和编译器中的新增功能](#)

描述新语言功能。

[Visual C# 2005 中的新增功能](#)

描述新的代码编辑器、开发环境、代码向导和调试功能。

[将 Visual C# 应用程序升级到 Visual Studio 2005](#)

描述将现有项目更新到 Microsoft Visual Studio 2005。

请参见

其他资源

[Visual Studio](#)

Visual C# 入门

以下主题可帮助您开始使用 Microsoft Visual C# 2005 开发应用程序。这些主题也将为您介绍 Microsoft Visual Studio 2005 以及 C# 语言 2.0 版本中的许多新功能。

本节内容

[Visual C# 文档路线图](#)

提供有关 Visual C# 文档内容的高级介绍。

[C# 语言和 .NET Framework 介绍](#)

概述 C# 语言和 .NET 平台。

[Visual C# 2005 中的新增功能](#)

Microsoft Visual C# 2005 中的新增内容。

[C# 2.0 语言和编译器中的新增功能](#)

C# 2.0 版中的新增内容。

[将 Visual C# 应用程序升级到 Visual Studio 2005](#)

将现有项目更新到 Microsoft Visual Studio 2005。

[创建您的第一个 C# 应用程序](#)

编写、编译和运行一个简单的 C# 应用程序。

[使用 C# 初学者工具包](#)

使用 C# 初学者工具包。

[其他帮助资源 \(Visual C#\)](#)

提供指向其他帮助资源的链接。

[如何实现 - C#](#)

提供指向相关主题的链接，这些主题演示如何执行各种特定任务。

相关章节

[使用 Visual C# IDE](#)

Visual C# 开发环境使用指南。

[迁移到 Visual C#](#)

将 Java 和 Visual J++ 应用程序转换为 Visual C#。

[用 Visual C# 编写应用程序](#)

提供使用 C# 和 .NET Framework 执行常见编程任务的概览，以及指向更多详细文档的链接。

[C# 编程指南](#)

提供了有关 C# 编程概念的信息，并且介绍如何使用 C# 执行各种任务。

[C# 参考](#)

提供有关 C# 关键字、运算符、预处理器指令、编译器开关以及编译器错误和警告的详细参考信息。

[Visual C# 示例](#)

演示如何使用 Visual C# 编程的源代码示例。

[C# 术语](#)

C# 术语词汇表。

请参见

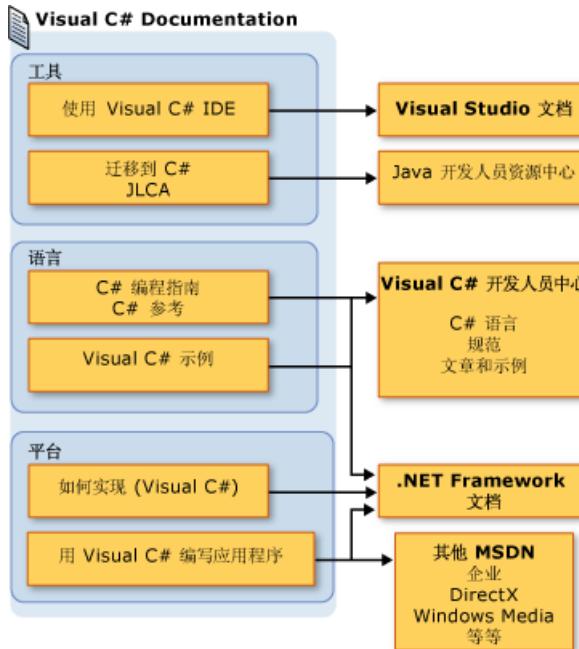
[其他资源](#)

Visual C#
Visual Studio

Visual C# 文档路线图

Microsoft Visual C# 2005 文档包含了特定于 C# 语言的信息，如关键字、编译器选项、错误信息和编程概念。此文档还向您提供了有关如何使用集成开发环境 (IDE) 的概述。此外，还有许多链接指向有关以下内容的更加详细的帮助：.NET Framework 类、ASP.NET Web 开发、调试、SQL 数据库编程以及更多。

下面的关系图为 Visual C# 文档内容，以及这些内容与 Visual Studio 文档和 MSDN Online 的其他相关章节的关系提供了一个概念视图。



到 C# 特定文档的快速链接

[Visual C# 入门](#)

[如何实现 - C#](#)

[迁移到 Visual C#](#)

[使用 Visual C# IDE](#)

[用 Visual C# 编写应用程序](#)

[C# 参考](#)

[Visual C# 示例](#)

[C# 编译器选项](#)

到相关文档的快速链接

有关创建 Windows 应用程序的更多信息，请参见[基于 Windows 的应用程序、组件和服务](#)。

有关创建 Web 应用程序的更多信息，请参见[Visual Web Developer](#)。

有关 .NET Framework 类库的更多信息，请参见[.NET Framework 类库概述](#)。

有关 .NET Framework 公共语言运行库、通用类型系统和其他相关概念的更多信息，请参见[.NET Framework 概述](#)。

[请参见](#)

[其他资源](#)

[Visual Studio](#)

C# 语言和 .NET Framework 介绍

C# 是一种简洁、类型安全的面向对象的语言，开发人员可以使用它来构建在 .NET Framework 上运行的各种安全、可靠的应用程序。使用 C#，您可以创建传统的 Windows 客户端应用程序、XML Web services、分布式组件、客户端 - 服务器应用程序、数据库应用程序以及很多其他类型的程序。Microsoft Visual C# 2005 提供高级代码编辑器、方便的用户界面设计器、集成调试器和许多其他工具，以在 C# 语言版本 2.0 和 .NET Framework 的基础上加快应用程序的开发。

注意

Visual C# 文档假设您了解基本的编程概念。如果您是初学者，可能需要学习一下 Visual C# 速成版，它可以从网站下载。您也可以利用任何关于 C# 的优秀书籍和 Web 资源来学习实用编程技巧。

C# 语言

C# 语法表现力强，只有不到 90 个关键字，而且简单易学。C# 的大括号语法使任何熟悉 C、C++ 或 Java 的人都可以立即上手。了解上述任何一种语言的开发人员通常在很短的时间内就可以开始使用 C# 高效地工作。C# 语法简化了 C++ 的诸多复杂性，同时提供了很多强大的功能，例如可为空的值类型、枚举、委托、匿名方法和直接内存访问，这些都是 Java 所不具备的。C# 还支持泛型方法和类型，从而提供了更出色的类型安全和性能。C# 还提供了迭代器，允许集合类的实现者定义自定义的迭代行为，简化了客户端代码对它的使用。

作为一种面向对象的语言，C# 支持封装、继承和多态性概念。所有的变量和方法，包括 `Main` 方法（应用程序的入口点），都封装在类定义中。类可能直接从一个父类继承，但它可以实现任意数量的接口。重写父类中的虚方法的各种方法要求 `override` 关键字作为一种避免意外重定义的方式。在 C# 中，结构类似于一个轻量类；它是一种堆栈分配的类型，可以实现接口，但不支持继承。

除了这些基本的面向对象的原理，C# 还通过几种创新的语言结构加快了软件组件的开发，其中包括：

- 封装的方法签名（称为委托），它实现了类型安全的事件通知。
- 属性（Property），充当私有成员变量的访问器。
- 属性（Attribute），提供关于运行时类型的声明性元数据。
- 内联 XML 文档注释。

在 C# 中，如果需要与其他 Windows 软件（如 COM 对象或本机 Win32 DLL）交互，可以通过一个称为“Interop”的过程来实现。互操作使 C# 程序能够完成本机 C++ 应用程序可以完成的几乎任何任务。在直接内存访问必不可少的情况下，C# 甚至支持指针和“不安全”代码的概念。

C# 的生成过程比 C 和 C++ 简单，比 Java 更为灵活。没有单独的头文件，也不要求按照特定顺序声明方法和类型。C# 源文件可以定义任意数量的类、结构、接口和事件。

下列各项是其他 C# 资源：

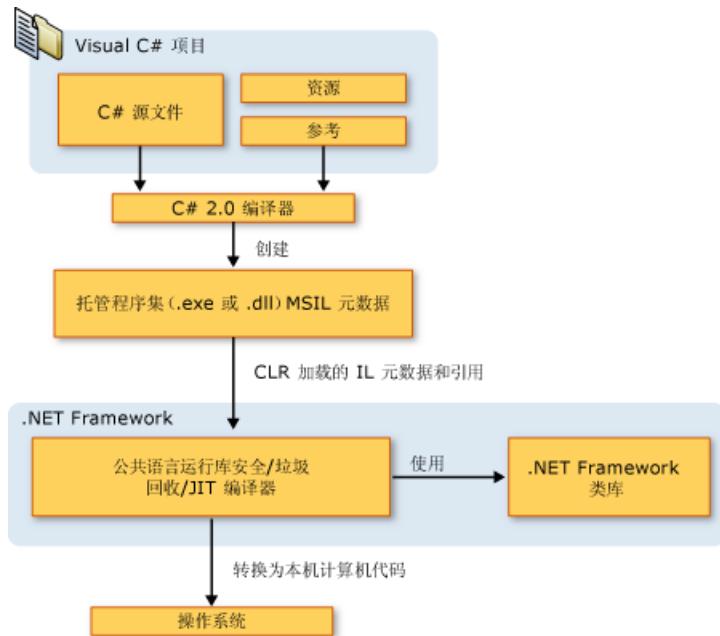
- 有关该语言的充分常规介绍，请参阅 [C# 语言规范](#) 的第 1 章。
- 有关 C# 语言特定方面的详细信息，请参阅 [C# 参考](#)。
- 有关 C# 语法与 Java 和 C++ 语法的比较，请参阅 [C# 编程语言（针对 Java 开发人员）](#) 和 [C#（针对 C++ 开发人员）](#)。

.NET Framework 平台体系结构

C# 程序在 .NET Framework 上运行，它是 Windows 的一个必要组件，包括一个称为公共语言运行时（CLR）的虚拟执行系统和一组统一的类库。CLR 是 Microsoft 的公共语言基础结构（CLI）的一个商业实现。CLI 是一种国际标准，是用于创建语言和库在其中无缝协同工作的执行和开发环境的基础。

用 C# 编写的源代码被编译为一种符合 CLI 规范的中间语言（IL）。IL 代码与资源（如位图和字符串）一起作为一种称为程序集的可执行文件存储在磁盘上，通常具有的扩展名为 .exe 或 .dll。程序集包含清单，它提供关于程序集的类型、版本、区域性和安全要求等信息。

执行 C# 程序时，程序集将加载到 CLR 中，这可能会根据清单中的信息执行不同的操作。然后，如果符合安全要求，CLR 执行实时（JIT）编译以将 IL 代码转换为本机机器指令。CLR 还提供与自动垃圾回收、异常处理和资源管理有关的其他服务。由 CLR 执行的代码有时称为“托管代码”，它与编译为面向特定系统的本机机器语言的“非托管代码”相对应。下图演示了 C# 源代码文件、基类库、程序集和 CLR 的编译时与运行时的关系。



语言互操作性是 .NET Framework 的一个关键功能。因为由 C# 编译器生成的 IL 代码符合公共类型规范 (CTS)，因此从 C# 生成的 IL 代码可以与从 Visual Basic、Visual C++、Visual J# 的 .NET 版本或者其他 20 多种符合 CTS 的语言中的任何一种生成的代码进行交互。单一程序集可能包含用不同 .NET 语言编写的多个模块，并且类型可以相互引用，就像它们是用同一种语言编写的。

除了运行时服务，.NET Framework 还包含一个由 4000 多个类组成的内容详尽的库，这些类被组织为命名空间，为从文件输入和输出到字符串操作、到 XML 分析、到 Windows 窗体控件的所有内容提供多种有用的功能。典型的 C# 应用程序使用 .NET Framework 类库广泛地处理常见的“日常”任务。

有关 .NET Framework 平台的更多信息，请参见 [.NET Framework 概述](#)。

请参见

[其他资源](#)

[Visual C#](#)

[用 Visual C# 编写应用程序](#)

Visual C# 2005 中的新增功能

Microsoft Visual C# 2005 包含在下列区域中的新功能：

- 语言和编译器
- 代码编辑器
- 开发环境
- 文档和语言规范
- 调试

语言和编译器

现在，C# 语言支持泛型类型、迭代器和分部类型。最新版本的 C# 编译器还包含新功能和新选项。有关更多信息，请参见 [C# 2.0 语言和编译器中的新增功能](#)。

代码编辑器

代码编辑器包含 Visual C# 2005 的下列新功能。

代码段

通过提供可填写的模板，代码段加速了通用代码构造的输入。代码段存储为 XML 文件，可轻松对其进行编辑和自定义。

- [代码段 \(C#\)](#)
- [如何: 使用代码段 \(C#\)](#)
- [如何: 使用外侧代码段](#)

重构

重构工具可自动重新构造源代码，例如，通过将局部变量升级为参数或将代码块转换为方法。

- [如何: 将局部变量提升为参数](#)
- [提取方法](#)
- [封装字段](#)
- [提取接口](#)
- [重命名](#)
- [移除参数](#)
- [重新排列参数](#)

开发环境

开发环境包括 Visual C# 2005 的下列增强功能。

IntelliSense

IntelliSense 得到了增强，具备下列新功能：

- 将光标退格到对象之前的范围运算符时，或者在撤消完成操作时，自动显示 [列出成员](#) 的完成列表。
- 当编写错误处理代码时，[列出成员](#) 通过从 `catch` 子句的完成列表中筛选不相关的成员，帮助您发现要捕捉哪些异常。
- 当需要插入标准化代码时，[自动代码生成](#) 现在允许您提示 IntelliSense 为您插入代码。
- IntelliSense 可用于创作 Web 应用程序。

类设计器

类设计器是一种新的、以图形方式显示类和类型的编辑器，并允许添加或修改方法。也可以从“类设计器”窗口中使用重构工具。

- 请参见 [设计和查看类与类型](#)。

对象测试工作台

对象测试工作台是为简单的对象级测试而设计的。它允许您创建对象的实例并调用其方法。

- 请参见[对象测试工作台](#)。

ClickOnce 部署

ClickOnce 部署允许您将 Windows 应用程序发布到 Web 服务器或网络文件共享上以简化安装。

- 请参见[ClickOnce 部署](#)。

强名称程序集的工具支持

“项目属性”对话框经过重新设计，它现在包括对签名程序集的支持。

- 请参见[项目属性](#)。

代码向导

下列代码向导现在已过时：

- C# 方法向导
- C# 属性向导
- C# 字段向导
- C# 索引器向导

文档和语言规范

C# 参考文档已经过大范围的重写，以提供开发人员在用 C# 创建应用程序时可能遇到的常见及高级用法问题的更全面信息。

C# 语言规范不再集成到帮助环境中，而是在两个 .doc 文件中提供。这些文件默认安装在 \\Microsoft Visual Studio 8\vcsharp\specifications\1033\之下。可从 MSDN 上的 C# 开发人员中心下载最新版本。有关更多信息，请参见[C# 语言规范](#)。

C# 专用的调试增强功能

已经增加了旨在为 C# 开发人员提供帮助的新功能，包括“编辑并继续”。

- 请参见[Visual Studio 2005 调试器的新增功能](#)。

请参见

概念

[Visual Studio 2005 中的新增功能](#)

其他资源

[Visual C#](#)

[Visual C# 入门](#)

[使用 Visual C# IDE](#)

C# 2.0 语言和编译器中的新增功能

随着 Visual Studio 2005 的发布, C# 语言已更新为 2.0 版, 它支持下列新功能:

泛型

该语言中添加了一些泛型类型, 使得程序员能够实现程度很高的代码重用, 获得更高的集合类性能。泛型类型只存在 arity 上的不同。也可以将参数强制为特定的类型。有关更多信息, 请参见[泛型类型参数](#)。

迭代器

迭代器使得规定 **foreach** 循环将如何循环访问集合的内容变得更加容易。

分部类

分部类型定义允许将单个类型(比如某个类)拆分为多个文件。Visual Studio 设计器使用此功能将它生成的代码与用户代码分离。

可空类型

可空类型允许变量包含未定义的值。在使用数据库和其他可能包含未含有具体值的元素的数据结构时, 可以使用可空类型。

匿名方法

现在, 可以将代码块作为参数来传递。在本应使用委托的任何地方, 都可以使用代码块来取代: 不需要定义新的方法。

命名空间别名限定符

命名空间别名限定符 (::) 对访问命名空间成员提供了更多控制。[global ::](#) 别名允许访问可能被代码中的实体隐藏的根命名空间。

静态类

若要声明那些包含不能实例化的静态方法的类, 静态类就是一种安全而便利的方式。C# 1.2 版要求将类构造函数定义为私有的, 以防止类被实例化。

外部程序集别名

通过 **extern** 关键字的这种扩展用法引用包含在同一程序集中的同一组件的不同版本。

属性访问器可访问性

现在可以为属性的 **get** 和 **set** 访问器定义不同级别的可访问性。

委托中的协变和逆变

现在传递给委托的方法在返回类型和参数方面可以具有更大的灵活性。

如何: 声明、实例化和使用委托

方法组转换为声明委托提供了一种更简单的语法。

固定大小的缓冲区

在不安全的代码块中, 现在可以声明包含嵌入数组的固定大小结构。

友元程序集

程序集可以提供对其他程序集的非公共类型的访问。

内联警告控制

#pragma 警告指令可用于禁用和启用某些编译器警告。

volatile

现在可以将 **volatile** 关键字应用于 **IntPtr** 和 **UIntPtr**。

此版本的 C# 编译器中引入了下列增加内容和更改内容:

/errorreport 选项

可用于通过 Internet 向 Microsoft 报告内部编译器错误。

/incremental 选项

已移除。

[/keycontainer](#) 和 [/keyfile](#) 选项

支持指定加密密钥。

[/langversion](#) 选项

可用于指定与特定语言版本的兼容性。

[/linkresource](#) 选项

包含附加选项。

[/moduleassemblyname](#) 选项

使您可以生成 .netmodule 文件并访问现有程序集中的非公共类型。

[/pdb](#) 选项

指定 .pdb 文件的名称和位置。

[/platform](#) 选项

使您可将 Itanium 系列 (IPF) 和 x64 结构作为目标平台。

[#pragma warning](#)

用于在代码中禁用或启用单个警告。

请参见

概念

[C# 编程指南](#)

其他资源

[C# 语言规范](#)

[C# 参考](#)

将 Visual C# 应用程序升级到 Visual Studio 2005

打开由 Visual Studio 早期版本创建的项目或解决方案文件时，升级向导会引导您逐步完成将项目转换为 Visual Studio 2005 的过程。升级向导可执行多种任务：创建新属性和新特性、删除过时属性和特性等。但是由于加强了错误检查功能，可能会遇到并非由编译器的早期版本生成的新的错误或警告消息。因此，升级现有应用程序的最后一步就是对代码进行更改，解决所有新错误。

在 C# 编译器的早期版本中生成一条消息的代码，在当前版本中经常会生成一条不同的消息。这通常是因为常规消息被更加具体的消息替换了。因为不需要更改代码，所以我们不记录此类差异。

由于错误检查更加严格，升级向导生成了以下新消息。

新的错误和警告消息

CS0121: 不明确的调用

因隐式转换的缘故，编译器无法调用重载方法的某种形式。可以用以下方法纠正该错误：

- 以不发生隐式转换的方式指定此方法的参数。
- 移除此方法的所有重载。
- 请在调用此方法之前，强制转换为正确的类型。

CS0122: 方法的保护级别导致无法访问方法

引用程序集（由 C++ 通过 `/d1PrivateNativeTypes` 编译器选项编译）中的类型时，可能会收到此错误。

由于在当前版本中，C++ 程序集生成的签名使用了未标记为公共的类型，发生此错误。

可以使用 `/test:AllowBadRetTypeAccess` 编译器选项解决此问题。此功能修复后，将移除该选项。

CS0429: 检测到无法访问的表达式代码

每当无法访问代码中某个表达式的一部分时，便会发生此错误。例如，条件 `false && myTest()` 满足此标准，这是因为该 `&&` 操作的左侧始终为 `false`，导致永远不会计算 `myTest()` 方法。若要修复此问题，请重复逻辑测试以消除无法访问的代码。

CS0441: 类不能既是静态的又是密封的

所有静态类也都是密封类。C# 语言规范禁止对一个类同时指定此两种修饰符，因此编译器现在将这种情况报告为错误。

若要修复此错误，请从该类中移除 `sealed`。

CS1699: 使用程序集签名属性时发出警告

指定签名的程序集属性已从代码移至编译器选项。现在，在代码中使用 `AssemblyKeyFile` 或 `AssemblyKeyName` 属性会产生此警告。

应使用以下编译器选项来代替这些属性：

- 使用 [/keyfile\(指定强名称密钥文件\)\(C# 编译器选项\)](#) 编译器选项代替 `AssemblyKeyFile` 属性，使用 [/keycontainer\(指定强名称密钥容器\)\(C# 编译器选项\)](#) 代替 `AssemblyKeyName`。

使用友元程序集时，不切换到命令行选项可能会阻碍编译器进行诊断。

如果使用的是 [/warnaserror\(将警告视为错误\)\(C# 编译器选项\)](#)，则可通过向编译器命令行添加 `/warnaserror-:1699` 将其重新转换为警告。如有需要，可以使用 `/nowarn:1699` 关闭此警告。

已移除增量编译

已移除 `/incremental` 编译器选项。编辑并继续功能取代了此功能。

请参见

其他资源

[C# 编译器选项](#)

创建您的第一个 C# 应用程序

只需花费几分钟，即可创建一个 C# 应用程序。按照下列步骤创建一个程序，该程序打开一个窗口并响应按钮按下操作。

过程

创建 C# 应用程序

1. 在“文件”菜单上，指向“新建”，然后单击“项目”。
2. 确保“Windows 应用程序”模板处于选中状态，在“名称”字段中，键入“MyProject”，然后单击“确定”。

在 Windows 窗体设计器中会显示一个 Windows 窗体。这是应用程序的用户界面。
3. 在“视图”菜单上，单击“工具箱”以使控件列表可见。
4. 展开“公共控件”列表，并将“标签”控件拖到您的窗体中。
5. 还要从工具箱“公共控件”列表中将一个按钮拖到窗体上靠近标签的位置。
6. 双击此新按钮以打开代码编辑器。Visual C# 已插入一个称为 button1_Click 的方法，单击该按钮时将执行此方法。
7. 将此方法更改为与以下类似：

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = "Hello, World!";
}
```

8. 按 F5 以编译并运行应用程序。

单击按钮时，将显示一条文本消息。祝贺您！您刚才已编写了自己的第一个 C# 应用程序。

请参见

其他资源

[Visual C#](#)

[Visual C# 入门](#)

Visual C# 版本中的项目模板

当创建新项目时，“新建项目”对话框和“新建网站”对话框中的图标表示可用的项目类型及其模板。您选择的项目模板确定了该项目的输出类型和可用于项目的其他选项。并非所有的项目模板在 Visual C# 的所有版本中都可用。

注意

对 Visual C# 速成版 或 Visual C# 标准版本中不可用的功能进行了介绍的文档可能包括在这些版本的文档集中。

Visual C# 项目模板

下表显示了在 Visual Studio 的不同版本中可用的 Visual C# 项目模板。

模板	Microsoft Visual C# 速成版	Visual Studio 2005 标准版	Visual Studio 2005 专业版及更高版本
ASP.NET 网站		X	X
ASP.NET Web 服务		X	X
ASP.NET Crystal Reports 网站			X
类库	X	X	X
控制台应用程序	X	X	X
Crystal Reports 应用程序			X
设备应用程序		X	X
空项目	X	X	X
空网站		X	X
Excel 模板			X
Excel 工作簿			X
初学者工具包 — 电影收藏	X	X	X
Outlook 外接程序			X
个人网站初学者工具包		X	X
Pocket PC 2003 : 类库		X	X
Pocket PC 2003 : 类库 (1.0)		X	X
Pocket PC 2003 : 控制台应用程序		X	X
Pocket PC 2003 : 控制台应用程序 (1.0)		X	X
Pocket PC 2003 : 控件库		X	X

Pocket PC 2003:设备应用程序		X	X
Pocket PC 2003:设备应用程序 (1.0)		X	X
Pocket PC 2003:空项目		X	X
Pocket PC 2003:空项目 (1.0)		X	X
初学者工具包 — 屏幕保护程序	X	X	X
SQL Server 项目			X
Smartphone 2003:类库 (1.0)		X	X
Smartphone 2003:控制台应用程序 (1.0)		X	X
Smartphone 2003:设备应用程序 (1.0)		X	X
Smartphone 2003:空项目 (1.0)		X	X
测试项目			X
Web 控件库		X	X
Windows 应用程序	X	X	X
Windows CE 5.0:类库		X	X
Windows CE 5.0:控制台应用程序		X	X
Windows CE 5.0:控件库		X	X
Windows CE 5.0:设备应用程序		X	X
Windows CE 5.0:空项目		X	X
Windows 控件库		X	X
Windows 服务			X
Word 文档			X
Word 模板			X

请参见

概念

[安装和设置要点](#)

[Visual Studio 2005 中的新增功能](#)

[Visual C# 2005 中的新增功能](#)

其他资源

[用 Visual C# 编写应用程序](#)

使用 C# 初学者工具包

初学者工具包是一个完整、独立的应用程序，可供您加载和生成。初学者工具包带有自己的文档，包括编程技巧的说明和如何对它进行自定义的建议。初学者工具包是了解实际运行的 C# 应用程序的一种良好途径。

加载并生成 Visual C# 初学者工具包

1. 在“文件”菜单上，单击“新建项目”。

出现“新建项目”对话框。此对话框列出 Visual C# 可以创建的不同的默认应用程序类型。

2. 选择一个“初学者工具包”应用程序类型，然后单击“确定”。

初学者工具包将加载到 Visual C#。

3. 要生成并启动初学者工具包项目，请按 F5。

请参见

其他资源

[Visual C#](#)

[Visual C# 入门](#)

其他帮助资源 (Visual C#)

下列站点和新闻组可帮助您查找常见和非常见问题的解答。

Microsoft 资源

下列网站由 Microsoft 维护，它们承载关于 C# 开发人员感兴趣的主題的文章和讨论组。

在 Web 上

[Microsoft 帮助与支持](#)

提供对知识库文章、下载和更新、支持 Web 广播以及其他服务的访问。

[Microsoft Visual C# 开发人员中心](#)

提供代码示例、升级信息以及技术内容。

[MSDN 讨论组](#)

提供了一个交流园地，您可以通过它与世界各地的专家进行交流。

论坛

[Microsoft 技术论坛](#)

为多项 Microsoft 技术(包括 C# 和 .NET Framework)建立的基于 Web 的论坛。

新闻组

[microsoft.public.dotnet.languages.csharp](#) [microsoft.public.dotnet.languages.csharp](#)

提供了一个论坛，供大家就 Visual C# 方面的问题进行提问和一般性讨论。

[microsoft.public.vsnet.general](#) [microsoft.public.vsnet.general](#)

提供了一个论坛，供大家讨论有关 Visual Studio 方面的问题和事项。

[microsoft.public.vsnet.ide](#) [microsoft.public.vsnet.ide](#)

提供了一个论坛，供大家讨论有关在 Visual Studio 环境下工作方面的问题。

[microsoft.public.vsnet.documentation](#) [microsoft.public.vsnet.documentation](#)

提供了一个论坛，供大家讨论有关 Visual C# 文档方面的问题和事项。

第三方资源

MSDN 的网站提供了当前相关的第三方站点和新闻组的信息。有关最新的可用资源列表，请参见 [MSDN 社区网站](#)。

请参见

其他资源

[Visual C#](#)

[Visual C# 入门](#)

[使用 Visual Studio 中的帮助](#)

[与其他开发人员交互](#)

[产品支持和辅助功能](#)

如何实现 - C#

"如何实现"是了解关于 C# 编程和应用程序开发的主要任务的主要主题的途径。此主题列出可以使用 C# 完成的基本任务类别。这些链接提供指向基于过程的重要"帮助"页面的指针。

C# 语言

[C# 语言规范](#)... [线程处理](#)... [泛型](#)... [代码示例片段](#)... [示例](#)... [更多](#)

.NET Framework

[文件 I/O](#)... [字符串](#)... [集合](#)... [序列化](#)... [组件](#)... [程序集和应用程序域](#)... [更多](#)

Windows 应用程序

[生成 Windows 应用程序](#)... [控件](#)... [Windows 窗体](#)... [绘图](#)... [更多](#)

网页和 Web 服务

[ASP.NET 网页](#)... [XML Web Services](#)... [更多](#)

调试

[使用 VS Debugger](#)... [.NET Framework Trace 类](#)... [调试 SQL 事务](#)... [更多](#)

数据访问

[连接到数据源](#)... [SQL Server](#)... [数据绑定](#)... [更多](#)

设计类

[类设计器](#)... [使用类和其他类型](#)... [创建和修改类型成员](#)... [类库设计准则](#)... [更多](#)

安全性

[代码访问安全性](#)... [安全策略最佳做法](#)... [权限集](#)... [更多](#)

Office 编程

[Office 编程](#)... [控件](#)... [Word](#)... [Excel](#)... [更多](#)

智能设备

[智能设备项目中的新增内容](#)... [智能设备编程](#)... [调试智能设备](#)... [更多](#)

部署

[ClickOnce](#)... [Windows Installer](#)

其他资源

访问以下网站需要 Internet 连接。

[Visual Studio 2005 开发人员中心](#)

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

[Visual C# 开发人员中心](#)

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

[Microsoft .NET Framework 开发人员中心](#)

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

请参见

其他资源

[Visual C# 入门](#)

C# 语言(如何实现 - C#)

此页面链接到广泛使用的 C# 语言任务的“帮助”。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

C# 语言

[C# 2.0 语言和编译器中的新增功能](#)

包含有关新功能的信息，包括泛型、迭代器、匿名方法和分部类型。

[使用 C# 初学者工具包](#)

解释如何加载并生成 Visual C# 初学者工具包。

[C# 语言规范](#)

提供指向规范的最新版本(Microsoft Word 格式)的链接。

命令行

[Main\(\) 和命令行参数\(C# 编程指南\)](#)

解释 Main 方法，它是程序的入口点，可以在此创建对象和调用其他方法。C# 程序中只能有一个入口点。

[如何: 使用 foreach 访问命令行参数\(C# 编程指南\)](#)

提供一个代码示例，显示如何访问命令行参数。

[如何: 显示命令行参数\(C# 编程指南\)](#)

解释如何通过 args 字符串数组显示命令行参数。

[Main\(\) 返回值\(C# 编程指南\)](#)

解释 Main 方法可能的返回值。

类与继承

[base\(C# 参考\)](#)

解释如何指定在创建派生类实例时调用的基类构造函数。

[如何: 了解向方法传递结构和向方法传递类引用之间的区别\(C# 编程指南\)](#)

包含一个代码示例，显示向方法传递结构时，将传递该结构的副本，而传递类实例时，将传递一个引用。

[实例构造函数\(C# 编程指南\)](#)

解释类构造函数与继承。

[如何: 编写复制构造函数\(C# 编程指南\)](#)

包含一个演示类的构造函数如何将另一对象看作参数的代码示例。

[如何: 在结构间实现用户定义的转换\(C# 编程指南\)](#)

包含一个定义了两个结构并演示二者之间的转换的代码示例。

数据类型

[装箱转换\(C# 编程指南\)](#)

包含一个阐释值类型和装箱对象能够存储不同的值的示例。

[取消装箱转换\(C# 编程指南\)](#)

包含一个阐释如何针对无效装箱情况显示错误消息的代码示例。

数组

[作为对象的数组\(C# 编程指南\)](#)

包含一个显示数组的维数的代码示例。

[交错数组\(C# 编程指南\)](#)

包含这样一个代码示例：它生成了一个数组，该数组的元素自身也是数组。

[将数组作为参数传递\(C# 编程指南\)](#)

包含这样一些代码示例：它们将一个字符串数组初始化，然后将其作为参数传递给 **PrintArray** 方法，该方法显示了该数组的元素。

[使用 `ref` 和 `out` 传递数组 \(C# 编程指南\)](#)

包含这样一些代码示例：它们演示了在将数组传递给方法的过程中使用 **out** 与 **ref** 时这二者的差异。

属性

[如何：声明和使用读/写属性 \(C# 编程指南\)](#)

包含一个示例，显示如何声明并使用读/写属性。

[如何：定义抽象属性 \(C# 编程指南\)](#)

包含一个代码示例，显示如何定义抽象属性。

方法

[传递值类型参数 \(C# 编程指南\)](#)

包含演示传递值类型的各种方法的代码示例。

[传递引用类型参数 \(C# 编程指南\)](#)

包含演示传递引用类型的各种方法的代码示例。

事件

[如何：订阅和取消订阅事件 \(C# 编程指南\)](#)

演示如何订阅由其他类发布的事件，包括窗体、按钮、列表框等等。

[如何：发布符合 .NET Framework 准则的事件 \(C# 编程指南\)](#)

演示如何创建基于 `EventHandler` 和 `EventHandler<T>` 的事件。

[如何：实现接口事件 \(C# 编程指南\)](#)

演示如何实现在接口中声明的事件。

[如何：使用字典存储事件实例 \(C# 编程指南\)](#)

解释如何使用哈希表存储事件实例。

[如何：引发派生类中的基类事件 \(C# 编程指南\)](#)

演示如何用受保护的虚方法包装基类事件，以便从派生类调用这些事件。

接口

[如何：显式实现接口成员 \(C# 编程指南\)](#)

显示如何声明显式实现接口的类，以及如何通过接口实例访问成员。

[如何：使用继承显式实现接口成员 \(C# 编程指南\)](#)

提供一个示例，同时以公制单位和英制单位显示框的尺寸。

泛型

[C# 泛型简介](#)

介绍如何通过泛型定义类型安全的集合类。您只实现一次泛型类，但可以使用任何类型来声明和使用它。

[.NET Framework 中的泛型](#)

解释 `System.Collections.Generic` 命名空间中新的泛型集合组的功能和使用方法。

[泛型代码中的默认关键字 \(C# 编程指南\)](#)

提供一个代码示例，演示如何使用类型参数的默认关键字。

[泛型方法 \(C# 编程指南\)](#)

介绍用于声明泛型方法的语法。还演示了在应用程序中使用泛型方法的示例。

[类型参数的约束 \(C# 编程指南\)](#)

演示如何约束类型参数，以启用对用于实例化泛型类的类型的方法和属性的访问。

[泛型委托 \(C# 编程指南\)](#)

包含用于声明泛型委托的语法。还包括有关实例化和使用泛型委托的重要备注、以及代码示例。

命名空间

[如何: 使用命名空间别名限定符 \(C# 编程指南\)](#)

讨论当与某一成员同名的另一个实体可能隐藏该成员时，在全局命名空间中访问该成员的能力。

迭代器

[如何: 为泛型列表创建迭代器块 \(C# 编程指南\)](#)

提供一个示例，其中使用整数数组生成 SampleCollection 列表。For 循环会遍历集合，并产生每一项的值。然后使用 **foreach** 循环显示集合中的项。

[如何: 为泛型列表创建迭代器块 \(C# 编程指南\)](#)

提供一个示例，其中泛型类 Stack<T> 实现泛型接口 IEnumator<T>。使用 Push 方法声明类型为 T 的数组并为其赋值。在 GetEnumerator 方法中，使用 yield return 语句返回数组的值。

委托

[如何: 合并委托 \(多路广播委托\) \(C# 编程指南\)](#)

提供一个示例，演示如何撰写多路广播委托。

[如何: 声明、实例化和使用委托 \(C# 编程指南\)](#)

提供一个示例，阐释如何声明、实例化和使用委托。

运算符重载

[如何: 使用运算符重载创建复数类 \(C# 编程指南\)](#)

显示如何使用运算符重载创建定义复数加法的复数类 Complex。

互操作性

[如何: 使用 COM Interop 进行 Word 拼写检查 \(C# 编程指南\)](#)

此示例阐释如何使用 C# 应用程序中的 Word 拼写检查工具。

[如何: 使用 COM Interop 创建 Excel 电子表格 \(C# 编程指南\)](#)

此示例阐释如何使用 .NET Framework COM 互操作功能在 C# 中打开现有的 Excel 电子表格。

[如何: 将托管代码用作 Excel 的自动化外接程序 \(C# 编程指南\)](#)

此示例阐释如何创建 C# 插件，以用于在 Excel 工作表的单元格中计算个人所得税率。

[如何: 使用平台调用播放波形文件 \(C# 编程指南\)](#)

此示例阐释如何使用平台调用服务，在 Windows 平台上播放波形声音文件。

不安全代码

[如何: 使用指针复制字节数组 \(C# 编程指南\)](#)

显示如何使用指针将字节从一个数组复制到另一个数组。

[如何: 使用 Windows ReadFile 函数 \(C# 编程指南\)](#)

显示如何调用 Windows ReadFile 函数，由于读取缓冲区要求将指针作为参数，因此该函数要求使用不安全上下文。

线程处理

[使用线程和线程处理](#)

提供一个主题列表，讨论托管线程的创建和管理，以及如何避免无法预料的后果。

[如何: 创建和终止线程 \(C# 编程指南\)](#)

提供一个示例，演示如何创建和启动线程，以及在同一进程中同时运行的两个线程如何交互。

[如何: 对制造者线程和使用者线程进行同步 \(C# 编程指南\)](#)

提供一个示例，显示如何使用 C# lock 关键字和 Monitor 对象的 Pulse 方法实现同步。

[如何: 使用线程池 \(C# 编程指南\)](#)

解释一个示例，该示例显示如何使用线程池。

字符串

[如何: 使用正则表达式搜索字符串 \(C# 编程指南\)](#)

解释如何使用可用的 Regex 类搜索字符串。这些搜索有的非常简单，有的复杂到需要完全使用正则表达式。

[如何: 联接多个字符串 \(C# 编程指南\)](#)

包含一个代码示例，演示如何联接多个字符串。

[如何: 使用字符串方法搜索字符串 \(C# 编程指南\)](#)

包含一个代码示例，演示如何使用 String 方法搜索字符串。

[如何: 使用 Split 方法分析字符串 \(C# 编程指南\)](#)

包含一个代码示例，演示如何使用 System.String.Split 方法分析字符串。

[如何: 修改字符串内容 \(C# 编程指南\)](#)

包含这样一个代码示例：它将字符串的内容提取到一个数组中，然后修改该数组的某些元素。

属性

[如何: 使用属性创建 C/C++ 联合 \(C# 编程指南\)](#)

包含一个示例，该示例使用 Serializable 属性将特定的特性应用到类中。

使用 DLL

[如何: 创建和使用 C# DLL \(C# 编程指南\)](#)

使用示例方案演示如何生成并使用 DLL。

程序集

[如何: 确定文件是否为程序集 \(C# 编程指南\)](#)

包含一个示例，该示例对某个 DLL 进行测试，以确定它是否为程序集。

[如何: 加载和卸载程序集 \(C# 编程指南\)](#)

解释如何在运行时将特定的程序集加载到当前的应用程序域中。

[如何: 与其他应用程序共享程序集 \(C# 编程指南\)](#)

解释如何与其他应用程序共享一个程序集。

应用程序域

[在另一个应用程序域中执行代码 \(C# 编程指南\)](#)

显示如何执行已加载到另一应用程序域中的程序集。

[如何: 创建和使用应用程序域 \(C# 编程指南\)](#)

演示如何使用运算符重载实现三值逻辑类型。

示例

[Visual C# 示例](#)

包含链接，用以打开或复制从 Hello World 示例到“泛型”示例 (C#) 范围内的示例文件。

请参见

概念

[如何实现 - C#](#)

.NET Framework(如何实现 - C#)

此页面链接到有关广泛使用的 .NET Framework 任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

常规

[C# 语言和 .NET Framework 介绍](#)

介绍 C# 语言与 .NET Framework 类库以及运行时执行引擎之间的关系。

[.NET Framework 概述](#)

提供对 .NET Framework 主要功能的概念性概述，包括公共语言运行库、.NET Framework 类库和跨语言互操作性。

[快速技术查找器](#)

提供对 .NET Framework 主要技术领域的快速参考。

文件 I/O

[如何: 创建目录列表](#)

创建一个新目录。

[如何: 对新建的数据文件进行读取和写入](#)

读取和写入新创建的数据文件。

[如何: 打开并追加到日志文件](#)

打开日志文件并在其中追加内容。

[如何: 向文件写入文本](#)

向文件写入文本。

[如何: 从文件读取文本](#)

从文件中读取文本。

[如何: 从字符串中读取字符](#)

从字符串中读取字符。

[如何: 向字符串写入字符](#)

向字符串写入字符。

[如何: 添加或移除访问控制列表项](#)

添加或移除访问控制列表 (ACL) 条目以增强安全性。

字符串

[创建新字符串](#)

如何创建新字符串。

[剪裁和移除字符](#)

如何在字符串的开头或结尾移除字符。

[填充字符串](#)

如何在字符串的开头或结尾添加制表符或空格。

[比较字符串](#)

如何比较两个字符串是否相等。

[更改大小写](#)

如何转换字母的大小写。

[使用 StringBuilder 类](#)

高效的字符串操作技巧。

如何:使用基本字符串操作执行字符串操作

如何拆分字符串, 如何向一个字符串追加另一个字符串, 等等。

如何:使用 System.Convert 转换数据类型

包含使用 Convert 类将字符串值转换为布尔值的示例。

如何:从字符串中剥离无效字符

包含使用静态 Regex.Replace 方法从字符串中清除无效字符的示例。

如何:验证字符串是否为有效的电子邮件格式

包含使用静态 Regex.IsMatch 方法验证字符串是否是有效的电子邮件格式的示例。

集合

集合和数据结构

概述 .NET Framework 集合类。

选择集合类

如何选择要使用的集合类型。

何时使用泛型集合

说明泛型集合类相对于非泛型集合类的优点。

System.Collections.Generic

介绍泛型集合类的门户页。

List

提供代码示例以演示如何在 `List<T>` 集合中添加和移除项。

SortedDictionary

提供代码示例以演示如何在 `SortedDictionary<K,V>` 集合中添加和移除键/值对。

异常

如何:在 Catch 块中使用特定异常

包含使用 **try/catch** 块捕捉 `InvalidCastException` 的示例。

如何:使用 Try/Catch 块捕捉异常

包含使用 **try/catch** 块捕捉可能的异常的示例。

如何:创建用户定义的异常

包含一个示例, 其中新的异常类 `EmployeeListNotFoundException` 从 `Exception` 派生而来。

如何:使用 Finally 块

包含使用 **try/catch** 块捕捉 `ArgumentOutOfRangeException` 异常的示例。

如何:显式引发异常

包含使用 **try/catch** 块捕捉可能的 `FileNotFoundException` 异常的示例。

事件

如何:在 Windows 窗体应用程序中使用事件

包含说明如何处理 Windows 窗体上的按钮单击事件的示例。

如何:将事件处理方法连接到事件

包含说明如何为事件添加事件处理方法的示例。

如何:引发和使用事件

包含使用在事件、委托和引发事件中详细描述的概念的示例。

如何:使用事件属性处理多个事件

包含说明如何使用事件属性处理多个事件的示例。

如何:在类中实现事件

包含描述如何在类中实现事件的步骤。

调试

请参见[调试\(如何实现 - C#\)](#)。

部署

请参见[安全性\(如何实现 - C#\)](#)。

服务组件

如何:创建补偿资源管理器 (CRM)

包括说明如何创建补偿资源管理器的代码示例

如何:创建服务组件

包含描述如何创建新服务组件的过程。

如何:将 Description 属性应用到程序集

说明如何应用 `DescriptionAttribute` 属性设置程序集的说明。

如何:使用 SetAbort 和 SetComplete 方法

演示如何使用 `ContextUtil` 类的 `SetComplete` 和 `SetAbort` 静态方法。

如何:将 ApplicationID 属性应用到程序集

演示如何将 `ApplicationID` 属性应用于程序集。

如何:创建池对象并设置其大小和超时限制

说明如何创建池对象并设置其大小和超时限制。

如何:创建使用自动事务的 Web 服务方法

描述如何创建使用自动事务的 Web 服务方法。

如何:为应用程序设置 SoapRoot 属性

演示如何将 `SoapVRoot` 属性设置为“`MyVRoot`”。

如何:设置事务超时

显示如何将事务超时设置为 10 秒。

如何:使用 ApplicationName 属性设置应用程序名称

演示如何通过使用程序集级别的 `ApplicationName` 属性提供应用程序名称。

如何:使用 COM+ 的 BYOT(带来您自己的事务)功能

包含说明从 `ServicedComponent` 类派生的类如何使用 COM+ 的 BYOT 功能访问分布式事务协调器 (DTC) 的步骤。

如何:创建专用组件

演示如何使用类的 `PrivateComponentAttribute` 属性。

如何:设置应用程序的激活类型

说明如何将激活类型设置为“`server`”。

如何:对类的实例启用同步

演示如何对 `TestSync` 类的实例启用同步。

如何:在 .NET Framework 类中使用自动事务

描述如何准备参与自动事务的类。

如何:启用 JIT 激活

说明如何在类内和类外启用和停用 JIT 激活。

如何:对事务识别类设置 AutoComplete 属性

演示如何在具有事务意识的类上放置 **AutoComplete** 属性。

如何:实现异步显示消息的排队组件

演示如何实现异步显示消息的排队组件。

如何:实现松耦合事件

包含一些步骤, 说明如何实现事件类和事件接收以及发行者以引发事件, 其中事件类和事件接收用于实现普通事件接口。

如何:配置对象构造

包含过程和示例, 描述如何配置对象结构并将 **TestObjectConstruct** 类的默认初始化字符串设置为“Initial Catalog=Northwind;Data Source=.\SQLServerInstance;Trusted_Connection=yes”字符串。

程序集和应用程序域

如何:从程序集获得类型和成员信息

包含从程序集中获取类型和成员信息的示例。

如何:生成单文件程序集

包含说明如何使用命令行编译器创建单文件程序集的过程。

如何:创建应用程序域

创建新的应用程序域, 并为它指定名称 MyDomain, 然后将宿主域和新创建的子应用程序域的名称打印到控制台。

如何:确定程序集的完全限定名

说明如何在控制台中显示包含指定类的程序集的完全限定名。

如何:配置应用程序域

创建 **AppDomainSetup** 类的实例, 用该类创建新的应用程序域, 将信息写入控制台, 然后卸载该应用程序域。

如何:查看程序集内容

包含一个示例, 此示例以基本的“Hello, World”程序开始, 并说明如何使用 Ilasm.exe 反汇编此 Hello.exe 程序集和查看程序集清单。

如何:引用具有强名称的程序集

创建称为 myAssembly.dll 的程序集, 此程序集从称为 myAssembly.cs 的代码模块中引用称为 myLibAssembly.dll 的具有强名称的程序集。

如何:卸载应用程序域

创建称为 MyDomain 的新应用程序域, 并将一些信息打印到控制台, 然后卸载该应用程序域。

如何:从全局程序集缓存中移除程序集

包含从全局程序集缓存中移除名为 hello.dll 的程序集的示例。

如何:将程序集安装到全局程序集缓存

包含将文件名为 hello.dll 的程序集安装到全局程序集缓存中的示例。

如何:生成多文件程序集

描述用于创建多文件程序集的过程, 并提供阐释该过程中每个步骤的完整示例。

如何:将程序集加载到应用程序域中

包含将程序集加载到当前应用程序域中, 然后执行该程序集的示例。

如何:使用强名称为程序集签名

包含一个示例, 该示例使用密钥文件 sgKey.snk, 用强名称对程序集 MyAssembly.dll 进行签名。

如何:查看全局程序集缓存的内容

说明如何使用全局程序集缓存工具 (Gacutil.exe) 查看全局程序集缓存的内容。

如何:创建公钥/私钥对

说明如何用强名称对程序集进行签名，以及如何使用强名称工具 (Sn.exe) 创建密钥对。

交互操作

如何:将类型库作为 Win32 资源嵌入基于 .NET 的应用程序

说明如何将类型库作为 Win32 资源嵌入基于 .NET Framework 的应用程序中。

如何:使用 Tlbimp.exe 生成主互操作程序集

提供使用 Tlbimp.exe 生成主互操作程序集的示例。

如何:手动创建主互操作程序集

提供手动创建主互操作程序集的示例。

如何:从类型库生成 Interop 程序集

提供从类型库生成互操作程序集的示例。

如何:引发 COM 接收器所处理的事件

提供一个示例，该示例将托管服务器显示为事件源，将 COM 客户端显示为事件接收器。

如何:自定义运行库可调用包装

说明如何通过修改 IDL 源或修改导入的程序集，自定义运行库可调用包装。

如何:对基于 .NET 的组件配置免注册激活

解释如何配置基于 .NET Framework 的组件以进行免注册激活。

如何:实现回调函数

演示托管应用程序如何才能使用平台调用打印本地计算机上每个窗口的句柄值。

如何:映射 HRESULT 和异常

包含一个示例，该示例创建名为 **NoAccessException** 的新异常类，并将其映射到 HRESULT E_ACCESSDENIED。

如何:编辑互操作程序集

演示如何以 Microsoft 中间语言 (MSIL) 指定封送处理更改。

如何:添加对类型库的引用

解释添加对类型库引用的步骤。

如何:处理 COM 源引发的事件

包含一个示例，该示例演示如何打开 Internet Explorer 窗口，并将由 **InternetExplorer** 对象引发的事件连接到在托管代码中实现的事件处理程序。

如何:手动创建包装

演示一个使用 IDL 编写的 **ISATest** 接口和 **SATest** 类的示例，以及 C# 源代码中的对应类型。

如何:注册主互操作程序集

包括注册 CompanyA.UtilLib.dll 主互操作程序集的示例。

如何:包装类型库的多个版本

解释如何包装类型库的多个版本。

安全性

请参见安全性(如何实现 - C#)。

序列化

如何:将对象反序列化

提供将对象反序列化到文件中的示例。

如何:使用 XML 架构定义工具生成类和 XML 架构文档

提供步骤，说明如何使用 XML 架构定义工具生成类和 XML 架构文档。

如何:为 XML 流指定一个备用元素名

说明如何用同一组类生成多个 XML 流。

[如何:控制派生类的序列化](#)

提供说明如何控制派生类的序列化的示例。

[如何:将对象序列化为 SOAP 编码的 XML 流](#)

提供将对象序列化为 SOAP 编码的 XML 流的过程和示例。

[如何:块化序列化数据](#)

提供实现服务器端分块和

客户端处理的过程和示例。

[如何:将对象序列化](#)

提供序列化对象的过程。

[如何:限定 XML 元素和 XML 属性名](#)

提供在 XML 文档中创建限定名的过程和示例。

[如何:重写已编码的 SOAP XML 序列化](#)

提供将对象的序列化作为 SOAP 消息重写的过程和示例。

[编码和本地化](#)

[如何:分析 Unicode 数字](#)

提供一个示例，该示例使用 **Decimal.Parse** 方法分析用于在不同脚本中指定数字的 Unicode 代码值的字符串。

[如何:创建自定义区域性](#)

提供定义和创建自定义区域性的过程。

[高级编程](#)

[如何:定义和执行动态方法](#)

说明如何定义和执行简单的动态方法和绑定到类实例的动态方法。

[如何:使用反射检查和实例化泛型类型](#)

提供步骤，说明如何发现和操作泛型类型。

[如何:定义具有反射发出的泛型方法](#)

提供步骤，说明如何用反射发出定义泛型方法。

[如何:使用完全签名来为动态程序集赋予强名称](#)

演示使用完全签名给动态程序集赋予一个强名称。

[如何:将程序集加载到仅反射上下文中](#)

提供过程和代码示例，说明如何将程序集加载到仅反射上下文中。

[如何:定义具有反射发出的泛型类型](#)

说明如何用两个类型参数创建简单的泛型类型，如何将类约束、接口约束和特殊约束应用于类型参数，以及如何创建将类的类型参数作为参数类型和返回类型使用的成员。

[.NET Framework 演练](#)

[演练:向 Windows 窗体组件添加智能标记](#)

演示如何使用一个简单的“ColorLabel”示例控件（从标准的 Windows 窗体“Label”控件派生）中的代码添加智能标记。

[演练:使用 SOAP 扩展更改 SOAP 消息](#)

说明如何生成和运行 SOAP 扩展。

[演练:使用 ASP.NET 生成基本的 XML Web 服务](#)

演示使用 ASP.NET 生成基本的 XML Web services。

[演练:针对特定设备自定义 ASP.NET 移动网页](#)

演示如何自定义特定设备。

[演练: 自定义服务描述和代理类的生成过程](#)

演示如何自定义服务说明和代理类的生成。

[演练: 手动部署 ClickOnce 应用程序](#)

描述使用命令行或清单生成和编辑工具 (Mage) 的图形化版本创建完全 ClickOnce 部署所需的步骤。

[演练: 使用 ClickOnce 部署 API 按需下载程序集](#)

演示如何将应用程序中的某些程序集标记为“可选的”，以及如何在公共语言运行库 (CLR) 要求这些程序集时使用 **System.Deployment.Application** 命名空间中的类来下载它们。

[演练: 实现 UI 类型编辑器](#)

解释如何为自定义类型创作自己的 UI 类型编辑器，以及如何通过使用 **PropertyGrid** 显示编辑界面。

其他资源

[Visual Studio 2005 开发人员中心](#)

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

[Visual C# 开发人员中心](#)

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

[Microsoft .NET Framework 开发人员中心](#)

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

请参见

[概念](#)

[如何实现 - C#](#)

Windows 应用程序 (如何实现 - C#)

此页面链接到广泛使用的 Windows 应用程序任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

常规

[基于 Windows 的应用程序概述](#)

提供可以使用 Microsoft Visual Studio 2005 创建的 Windows 应用程序的概述。

[在 Windows 窗体和 Web 窗体之间作出抉择](#)

讨论每种技术的功能和特性，帮助您确定最适合您的应用程序的技术。

使用窗体

[Windows 窗体设计器](#)

[如何: 在 Windows 应用程序中选择启动窗体](#)

提供如何在 Windows 应用程序中设置启动窗体的信息。

[如何: 将多个事件连接到 Windows 窗体中的单个事件处理程序](#)

解释在 Windows 窗体应用程序中，如何使用 C# 中“属性”窗口的“事件”视图，将多个事件连接到单个事件处理程序。

[如何: 用 Windows 窗体创建多窗格用户界面](#)

解释如何创建多窗格用户界面，它类似于 Microsoft Outlook 中使用的用户界面，带有“文件夹”列表、“邮件”窗格和“预览”窗格。

[如何: 向 Windows 窗体添加背景图像](#)

解释如何将背景图像置于控件或窗体本身。使用“属性”窗口，可以轻松完成此操作。

[如何: 在设计时为 Windows 窗体上的控件设置工具提示](#)

解释如何在代码或 Windows 窗体设计器中设置工具提示字符串。

[如何: 向 Windows 窗体添加 ActiveX 控件](#)

解释如何在 Windows 窗体上添加 ActiveX 控件。

[如何: 创建 Windows 窗体控件的访问键](#)

解释如何在菜单、菜单项或控件（如按钮）标签的文本中创建访问键。

[在运行时使用 Windows 窗体](#)

[如何: 在运行时在控件集合中进行添加或移除](#)

提供应用程序开发过程中的常见任务，例如将控件添加到窗体上任意容器控件中，或从窗体上的任意容器控件中移除控件。

[如何: 启用 Windows XP 视觉样式](#)

显示如何在 Windows 窗体的工作区中启用视觉样式。

[如何: 使启动 Windows 窗体不可见](#)

显示当应用程序启动时，如何使基于 Windows 的应用程序的主窗体不可见。

[如何: 将 Windows 窗体保持在最前面](#)

显示在设计时或编程时，如何使窗体成为位于 Windows 窗体应用程序最顶端的窗体。

[如何: 显示有模式和无模式 Windows 窗体](#)

显示如何将窗体显示为模式对话框或无模式对话框。

控件

[TextBox 控件](#)

[如何: 在 Windows 窗体 TextBox 控件中选择文本](#)

显示如何以编程方式在 Windows 窗体 TextBox 控件中选择文本。

[如何: 在字符串中放置引号 \(Windows 窗体\)](#)

显示如何将引号 (" ") 置于文本字符串中。

[如何: 创建只读文本框\(Windows 窗体\)](#)

显示如何将可编辑的 Windows 窗体文本框转换成只读控件。

[如何: 使用 Windows 窗体 TextBox 控件创建密码文本框](#)

显示如何使用 Windows 窗体 TextBox 控件创建密码文本框。

[如何: 控制 Windows 窗体 TextBox 控件中的插入点](#)

显示如何在 TextBox 控件中控制插入点。

[如何: 将数据绑定到 MaskedTextBox 控件](#)

显示如何将数据绑定到 MaskedTextBox 控件。

[演练: 使用 MaskedTextBox 控件](#)

阐释如何执行以下任务:

初始化 MaskedTextBox 控件。

当字符与掩码不符时警告用户。

当用户试图提交的值对该类型无效时警告用户。

RichTextBox 控件

[如何: 将文件加载到 Windows 窗体 RichTextBox 控件中](#)

解释如何将文件加载到 Windows 窗体 RichTextBox 控件(该控件可以显示纯文本、Unicode 纯文本或 RTF 文件)。

[如何: 在 Windows 窗体 RichTextBox 控件中显示滚动条](#)

提供 RichTextBox 控件中 ScrollBars 属性的七个可能的值, 在下表中对这些值进行说明。

[如何: 为 Windows 窗体 RichTextBox 控件设置字体属性](#)

描述如何使用 SelectionFont 属性使选定的字符变为粗体、带下划线或斜体格式。

[如何: 在 Windows 窗体 RichTextBox 控件中设置缩进、悬挂缩进和带项目符号的段落](#)

描述如何通过设置 SelectionBullet 属性将选定的段落设置为项目符号列表格式。也可以使用 SelectionIndent、SelectionRightIndent 和 SelectionHangingIndent 属性设置相对于控件的左右边缘以及其他文本行左边界的段落缩进。

[如何: 在 Windows 窗体 RichTextBox 控件中启用拖放操作](#)

描述如何通过处理 DragEnter 和 DragDrop 事件, 启用 Windows 窗体 RichTextBox 控件的拖放操作。

[如何: 使用 Windows 窗体 RichTextBox 控件显示 Web 样式的链接](#)

描述如何编写代码, 使单击链接时打开浏览器窗口, 该窗口显示链接文本指定的网站。

Button 控件

[如何: 响应 Windows 窗体按钮的单击](#)

解释 Windows 窗体按钮控件的最基本用法, 就是在单击按钮时执行某些代码。

[如何: 使用设计器将 Windows 窗体按钮指定为“接受”按钮](#)

解释如何将某个 Button 控件指定为接受按钮(也称作默认按钮)。每当用户按 Enter 键时, 即单击默认按钮, 而不管窗体上其他哪个控件具有焦点。

[如何: 使用设计器将 Windows 窗体按钮指定为“取消”按钮](#)

解释如何将某个 Button 控件指定为取消按钮。每当用户按 Esc 键时, 即单击取消按钮, 而不管窗体上的其他哪个控件具有焦点。通常设计这样的按钮是为了使用户可以快速退出操作而无须执行任何操作。

CheckBox 控件

[如何: 使用 Windows 窗体 CheckBox 控件设置选项](#)

提供有关如何使用 Windows 窗体 CheckBox 控件为用户提供真/假或是/否选项的信息。当控件选定时, 将显示一个复选标记。

[如何: 响应 Windows 窗体 CheckBox 的单击](#)

解释如何编写应用程序，以根据复选框的状态执行某些操作。

RadioButton 控件

[如何:按功能分组 Windows 窗体 RadioButton 控件](#)

解释如何通过将单选按钮拖到容器(如面板控件、GroupBox 控件或窗体)内部以将它们分组。

ListBox 控件、ComboBox 控件和 CheckedListBox 控件

[如何:将 Windows 窗体 ComboBox 控件或 ListBox 控件绑定到数据](#)

解释如何将 ComboBox 和 ListBox 与数据绑定以执行任务，如浏览数据库中的数据、输入新数据、或编辑现有数据。

[如何:为 Windows 窗体 ComboBox 控件、ListBox 控件或 CheckedListBox 控件创建查找表](#)

提供表格，显示如何存储和显示食品订货单数据的示例。

[如何:在 Windows 窗体 ComboBox 控件、ListBox 控件或 CheckedListBox 控件中添加或移除项](#)

提供如何将项添加到 Windows 窗体组合框、列表框或复选列表框的示例。但是，本主题演示最简单的方法，并且不需要数据绑定。

[如何:访问 Windows 窗体 ComboBox 控件、ListBox 控件或 CheckedListBox 控件中的特定项](#)

演示如何访问 Windows 窗体组合框、列表框或复选列表框中的特定项。它使您可以编程方式确定列表中任意给定位置的内容。

[如何:对 Windows 窗体 ComboBox 控件、ListBox 控件或 CheckedListBox 控件的内容排序](#)

演示如何使用支持排序的数据源：数据视图、数据视图管理器和已排序数组。

CheckedListBox 控件

[如何:确定 Windows 窗体 CheckedListBox 控件中的选定项](#)

演示如何通过循环访问存储于 CheckedItems 属性中的集合，或者使用 GetItemChecked 方法逐项通过列表，确定 Windows 窗体 CheckedListBox 控件中已选中的项。

DataGridView 控件

[如何:使用设计器将数据绑定到 Windows 窗体的 DataGridView 控件](#)

解释如何使用设计器将 DataGridView 控件连接到几种不同类型的数据源(包括数据库、业务对象或 Web 服务)。

[如何:验证 Windows 窗体 DataGridView 控件中的数据](#)

演示如何验证用户输入 DataGridView 控件的数据。

[如何:处理在 Windows 窗体 DataGridView 控件中输入数据时发生的错误](#)

演示如何使用 DataGridView 控件向用户报告数据输入错误。

[如何:为 Windows 窗体 DataGridView 控件中的新行指定默认值](#)

演示如何使用 DefaultValuesNeeded 事件为新行指定默认值。

[如何:创建未绑定的 Windows 窗体 DataGridView 控件](#)

演示如何以编程方式填充 DataGridView 控件，而无需将它绑定到数据源。

[如何:将未绑定的列添加到绑定了数据的 Windows 窗体 DataGridView 控件](#)

演示如何创建“详细信息”按钮的未绑定列，以便在实现主控/详细方案时，显示与父表中特定行相关的子表。

[如何:在 Windows 窗体 DataGridView 控件的单元格中显示图像](#)

演示如何从嵌入的资源中提取图标，并将它转换成位图，以便在图像列的每个单元格中显示。

[如何:在 Windows 窗体 DataGridView 单元格中承载控件](#)

显示如何创建日历列。此列的单元格按普通文本框单元格的形式显示日期，但是当用户编辑单元格时，将出现 DateTimePicker 控件。

[演练:验证 Windows 窗体 DataGridView 控件中的数据](#)

演示如何从 Northwind 示例数据库中的 Customers 表中检索行，然后在 DataGridView 控件中进行显示。当在 CompanyName 列编辑单元格时，如果试图离开单元格，将检查此新公司名称字符串以确保其不为空；如果新值是一个空字符串，DataGridView 将阻止光标离开该单元格，直到输入一个非空字符串。

演练:处理在 Windows 窗体 DataGridView 控件中输入数据时发生的错误

演示如何从 Northwind 示例数据库中的 Customers 表中检索行, 然后在 DataGridView 控件中进行显示。当在新行或已编辑的现有行中检测到重复的 CustomerID 值时, 将发生 DataError 事件, 将通过显示 MessageBox(显示此异常)处理此事件。

演练:创建未绑定的 Windows 窗体 DataGridView 控件

显示如何填充 DataGridView 控件, 以及如何在“未绑定”模式下管理行的添加和删除。

DataGridView 布局和格式设置

如何:使用设计器使 Windows 窗体 DataGridView 控件中的列成为只读

演示如何使包含数据的列变为只读的过程。

如何:使用设计器在 Windows 窗体的 DataGridView 控件中启用列重新排序

演示如何允许用户对列重新排序。启用列重新排序后, 用户可以使用鼠标拖动列标头, 将其移至新位置。

如何:使用设计器更改 Windows 窗体 DataGridView 控件中列的顺序

解释如何使用设计器在 Windows 窗体 DataGridView 控件中更改列的顺序。

如何:使用设计器添加和移除 Windows 窗体 DataGridView 控件中的列

解释如何使用设计器在 Windows 窗体 DataGridView 控件中添加或移除列。

使用控件进行数据绑定

如何:处理因数据绑定而发生的错误和异常

演示如何处理数据绑定操作时发生的错误和异常。

BindingSource 控件

如何:使用设计器将 Windows 窗体控件与 BindingSource 组件进行绑定

演示在设计时如何绑定控件。

如何:使用 Windows 窗体 BindingSource 组件创建查找表

演示如何使用 ComboBox 控件显示具有从父表到子表的外键关系的字段。

如何:使用 BindingSource 在 Windows 窗体控件中反映数据源更新

演示如何使用 ResetBindings 方法通知绑定控件有关数据源中的更新。

如何:使用 Windows 窗体 BindingSource 组件对 ADO.NET 数据进行排序和筛选

演示如何使用 BindingSource 对数据进行排序和筛选。

如何:使用 Windows 窗体 BindingSource 绑定到 Web 服务

演示如何创建和绑定到客户端代理。

绑定导航器

如何:使用 Windows 窗体 BindingNavigator 控件定位数据

解释如何设置 BindingNavigator 控件。

如何:使用 Windows 窗体 BindingNavigator 控件浏览数据集

演示如何使用 BindingNavigator 控件浏览数据库查询的结果。

ListView

如何:使用 Windows 窗体 ListView 控件添加和移除项

解释将项添加到 Windows 窗体 ListView 控件或从中移除的过程。可在任何时候添加或移除列表项。

如何:向 ListView 控件添加搜索功能

演示如何在短时间内创建具有专业外观的 Windows 窗体应用程序。

如何:选择 Windows 窗体 ListView 控件中的项

演示如何用编程方式在 Windows 窗体 ListView 控件中选择项。

如何:显示 Windows 窗体 ListView 控件的图标

演示如何在列表视图中显示图像。

[如何: 使用 Windows 窗体 ListView 控件在列中显示子项](#)

演示如何将子项添加到列表项中。

TreeView

[如何: 设置 Windows 窗体 TreeView 控件的图标](#)

演示如何在树视图中显示图像。

[如何: 添加和删除 Windows 窗体 TreeView 控件中的节点](#)

演示如何以编程方式将节点添加到树视图中或从中移除。

[如何: 确定被单击的 TreeView 节点 \(Windows 窗体\)](#)

演示如何确定单击了哪个 TreeView 节点。

容器控件

[如何: 水平拆分窗口](#)

解释如何制作水平拆分 SplitContainer 控件的拆分器。

[如何: 用 Windows 窗体创建多窗格用户界面](#)

演示如何创建一个多窗格用户界面，它类似于 Microsoft Outlook 中使用的用户界面，具有“文件夹”列表、“邮件”窗格和“预览”窗格。

[如何: 在 TableLayoutPanel 控件中跨行和跨列](#)

演示 TableLayoutPanel 控件中的控件如何跨越相邻的行和列。

[演练: 使用 TableLayoutPanel 在 Windows 窗体上排列控件](#)

阐释如何执行以下任务：

创建 Windows 窗体项目。

按行和列排列控件。

设置行属性和列属性。

使用控件跨行和跨列。

自动处理溢出。

通过在工具箱中双击控件插入控件。

通过绘制控件的轮廓插入控件。

将现有控件重新分配到不同的父级。

[演练: 使用 FlowLayoutPanel 在 Windows 窗体上排列控件](#)

阐释如何执行以下任务：

创建 Windows 窗体项目。

水平和垂直排列控件。

更改流方向。

插入流中断符。

使用空白和边距排列控件。

通过在工具箱中双击控件插入控件。

通过绘制控件的轮廓插入控件。

使用插入符号插入控件。

将现有控件重新分配到不同的父级。

图片控件和图像控件

[如何: 使用设计器加载图片 \(Windows 窗体\)](#)

解释在设计时，如何将 Image 属性设置为有效图片，从而在窗体中加载和显示图片。

如何:在运行时设置图片(Windows 窗体)

解释如何以编程方式设置 Windows 窗体 PictureBox 控件显示的图像。

如何:在运行时修改图片的大小或位置(Windows 窗体)

解释如何将窗体中的 Windows 窗体 PictureBox 控件的SizeMode 属性设置为不同值

DateTimePicker

如何:使用 Windows 窗体 DateTimePicker 控件设置和返回日期

解释如何在显示控件前设置 Value 属性, 以确定控件中最初选定的日期。

如何:使用 Windows 窗体 DateTimePicker 控件以自定义格式显示日期

解释如何显示自定义格式, 并将 CustomFormat 属性设置为格式字符串。

MonthCalendar

如何:在 Windows 窗体 MonthCalendar 控件中选择日期范围

演示如何使用 MonthCalendar 控件的属性设置日期范围, 或获取由用户设置的选择范围。

如何:使用 Windows 窗体 MonthCalendar 控件以粗体显示特定日期

演示如何以粗体或普通字体显示日期。

如何:在 Windows 窗体 MonthCalendar 控件中显示多个月份

演示如何在 Windows 窗体 MonthCalendar 控件中显示多个月份。

如何:更改 Windows 窗体 MonthCalendar 控件的外观

演示如何更改月历的配色方案, 如何在控件底部显示当前日期以及如何显示周数。

数据访问(针对 Windows 窗体)

演练:在 Windows 应用程序的不同窗体间传递数据

提供将数据从一个窗体传递到另一个窗体上的方法的分步介绍。

演练:在 Windows 应用程序中的窗体上显示数据

创建一个简单的窗体, 在多个单独的控件中显示单个表中的数据。

演练:在 Windows 应用程序中创建一个用于搜索数据的窗体

演示如何创建搜索数据的 Windows 窗体。

ToolStrip

如何:将 ToolStripItem 置于 ToolStrip

解释如何将 ToolStripItem 移至或添加到 ToolStrip 的左侧或右侧。

如何:使用设计器禁用 ToolStripMenuItem

解释如何在设计时禁用菜单项。

如何:移动 ToolStripMenuItem

解释如何将整个顶级菜单及其菜单项移动到 MenuStrip 上的其他位置。也可以在顶级菜单之间移动各个菜单项, 或在菜单内更改菜单项的位置。

如何:在 Windows 窗体中更改 ToolStrip 文本和图像的外观

解释如何控制是否在 ToolStripItem 中显示文本和图像, 以及它们之间如何对齐、相对于 ToolStrip 如何对齐。

上下文菜单

如何:将快捷菜单与 Windows 窗体 NotifyIcon 组件关联

演示如何将快捷菜单与 Windows 窗体 NotifyIcon 组件相关联。

如何:使用 Windows 窗体 ContextMenu 组件添加和移除菜单项

说明如何在 Windows 窗体中添加和移除快捷菜单项。

打印

如何:创建标准的 Windows 窗体打印作业

显示如何通过编写代码处理 PrintPage 事件，指定打印内容和打印方式。

如何:完成 Windows 窗体打印作业

显示如何通过处理 PrintDocument 组件的 EndPrint 事件来完成打印作业。

如何:打印 Windows 窗体中的多页文本文件

显示如何使用将对象(图形或文本)拖动到设备(如屏幕或打印机)的方法在 Windows 窗体中打印文本。

如何:在 Windows 窗体中选择连接到用户计算机的打印机

显示如何选择打印机，然后打印文件。

如何:在运行时从 PrintDialog 中捕获用户输入

显示在运行时如何更改打印选项。通过 PrintDialog 组件和 PrinterSettings 类可完成此项操作。

用户控件和自定义控件

向用户控件添加控件

演示如何将控件添加到用户控件。

向用户控件添加代码

演示如何将代码添加到用户控件。

多文档界面 (MDI)

如何:创建 MDI 父窗体

演示如何在设计时创建 MDI 父窗体。

如何:创建 MDI 子窗体

演示如何创建显示 RichTextBox 控件的 MDI 子窗体，该窗体与大多数文字处理应用程序相似。

如何:排列 MDI 子窗体

演示如何将子窗体显示为层叠、水平平铺或垂直平铺，或者显示为排列在 MDI 窗体下部的子窗体图标。

如何:确定活动的 MDI 子窗体

演示如何确定活动 MDI 子窗体，并将它的文本复制到剪贴板。

如何:将数据发送到活动的 MDI 子窗体

演示如何将数据从剪贴板发送到活动 MDI 子窗口。

图形

如何:绘制空心形状

显示如何在窗体上绘制空心椭圆和矩形。

如何:创建线性渐变

显示如何使用水平线性渐变画笔填充线条、椭圆和矩形。

如何:创建路径渐变

显示如何自定义用渐变颜色填充形状的方式。

如何:使用线条、曲线和形状创建图形

解释如何创建具有单个或多个图形的路径。

如何:创建用于绘制的 Graphics 对象

解释如何创建绘制的图形对象。

如何:创建缩略图像

演示如何由位图文件构造图像对象。

如何:创建垂直文本

演示如何使用 StringFormat 对象指定在垂直方向而不是在水平方向绘制文本。

如何:对齐绘制的文本

演示如何在矩形内绘制文本。每行文本都居中，整个文本块在矩形内居中。

[如何:在 Windows 窗体上绘制线条](#)

演示如何在窗体上绘制线条。

[如何:旋转、反射和扭曲图像](#)

演示如何通过指定原始图像的左上角、右上角和左下角作为目标点对图像进行旋转、反射和扭曲。

[如何:在 Windows 窗体上绘制文本](#)

显示如何使用 Graphics 的 DrawString 方法在窗体中绘制文本。

[如何:加载和显示位图](#)

显示如何从文件加载位图，并在屏幕上显示该位图。

[如何:加载和显示图元文件](#)

显示如何使用 Metafile 类的方法记录、显示和检查矢量图像。

[本地化和全球化 Windows 窗体](#)

[演练:本地化 Windows 窗体](#)

演示本地化 Windows 应用程序项目的过程。

[如何:使用 AutoSize 属性和 TableLayoutPanel 控件支持对 Windows 窗体的本地化](#)

演示如何启用适应不同字符串大小的布局。

[如何:为 Windows 窗体全球化设置区域性和用户界面的区域性](#)

演示如何设置适合于特定区域性的格式设置选项。

[如何:为全球化在 Windows 窗体中按从右到左的顺序显示文本](#)

演示如何从右向左显示文本。

[其他资源](#)

[Visual Studio 2005 开发人员中心](#)

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

[Visual C# 开发人员中心](#)

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

[Microsoft .NET Framework 开发人员中心](#)

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

[请参见](#)

[概念](#)

[如何实现 - C#](#)

网页和 Web 服务(如何实现 - C#)

本页面链接到广泛使用的 Web 应用程序任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

网页

[Visual Studio Web 开发的新增功能](#)

介绍用于创建 ASP.NET 网页的开发工具 Visual Web Developer。

[ASP.NET 网页介绍](#)

提供 ASP.NET 网页在 Web 应用程序中运行方式基本特征的概述。

Web 服务

[XML 文档和数据](#)

提供指向一些文章的链接，这些文章介绍如何读写 XML 文档和数据。

[在托管代码中使用的 XML Web services](#)

提供指向一些文章的链接，这些文章介绍如何创建和部署 XML Web services 以及如何使用托管代码访问 XML Web services。

[Visual Basic 和 Visual C# 中的 XML Web services 入门](#)

描述 Visual Studio 和 XML Web services 如何提供简单、灵活、基于标准的模型，该模型允许开发人员组装应用程序，无需考虑平台、编程语言或对象模型。

[如何：使用托管代码访问 XML Web services](#)

介绍如何访问 Web 服务。

[如何：使用托管代码异步访问 XML Web services](#)

介绍异步访问 Web 服务的方法。

[如何：从浏览器访问 XML Web 服务](#)

介绍如何从浏览器访问 XML Web services。

[如何：浏览 XML Web services 的内容](#)

介绍如何探索 XML Web services 的内容。

[如何：创建 ASP.NET Web 服务项目](#)

介绍如何创建 ASP.NET Web 服务项目。

[如何：使用 WebService 属性](#)

介绍如何使用 WebService 属性指定 XML Web services 的命名空间和说明文本。

[如何：从 WebService 类继承](#)

介绍如何从 WebService 类继承。

[如何：创建 XML Web services 方法](#)

介绍如何创建 XML Web services 方法。

[如何：使用 WebMethod 属性](#)

介绍如何使用 WebMethod 属性指明希望将方法公开为 XML Web services 的一部分。

[如何：使用托管代码调试 XML Web services](#)

介绍如何调试使用托管代码编写的 XML Web services。

[如何：使用托管代码部署 XML Web services](#)

介绍如何部署用托管代码编写的 XML Web services。

[如何：生成 XML Web services 代理](#)

介绍如何生成 XML Web services 代理。

[如何: 创建控制台应用程序客户端](#)

介绍如何创建控制台应用程序形式的 Web 服务客户端。

[如何: 处理由 Web 服务方法引发的异常](#)

介绍如何处理由 Web 服务方法引发的异常。

[如何: 使用回调方法实现异步 Web 服务客户端](#)

介绍如何使用回调技术实现异步 Web 服务客户端。

[如何: 使用等待方法实现异步 Web 服务客户端](#)

介绍如何使用等待技术实现异步 Web 服务客户端。

[演练: 安装时重定向应用程序以面向另一个 XML Web services](#)

说明如何使用 URL Behavior 属性、Installer 类和 Web 安装项目，创建可重定向以面向不同的 XML Web services 的 Web 应用程序。

[保证使用 ASP.NET 创建的 XML Web 服务的安全](#)

总结使用 ASP.NET 生成的 Web 服务可用的身份验证和授权选项。

[其他资源](#)

访问以下站点需要 Internet 连接。

[MSDN Web 服务和其他分布式技术开发人员中心](#)

提供有关开发和连接到 Web 服务的文章、代码示例以及其他许多资源。

[MSDN XML 开发人员中心](#)

提供有关处理 XML 数据的文章、代码示例以及其他许多资源。

[Microsoft ASP.NET 开发人员中心](#)

提供有关创建 ASP.NET 网页的文章、代码示例以及其他许多资源。

[请参见](#)

[概念](#)

[如何实现 - C#](#)

调试(如何实现 - C#)

此页面链接到广泛使用的调试任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

使用 Visual Studio 调试器 在 Visual Studio 中生成

讨论生成应用程序时进行连续测试和调试的工具。

使用 Visual Studio 进行调试

讨论使用 Visual Studio 调试器的基本知识。

调试器指南

提供指向一些文章的链接，这些文章介绍了基本调试任务以及调试器的功能。

.NET Framework 的跟踪功能

[如何:向应用程序代码添加跟踪语句](#)

解释如何使用以下方法：Write、Writelf、WriteLine、WriteLinelf、Assert 和 Fail，在应用程序中进行跟踪。

[如何:创建和初始化跟踪侦听器](#)

解释如何创建和初始化跟踪侦听器。

[如何:将 TraceSource 和筛选器与跟踪侦听器一起使用](#)

描述如何结合应用程序配置文件使用 TraceSource。

[如何:创建和初始化跟踪开关](#)

解释如何创建和初始化跟踪开关。

[如何:使用跟踪和调试进行条件编译](#)

解释如何以多种方式指定应用程序的编译器设置。

[如何:创建和初始化跟踪源](#)

解释如何使用配置文件协助跟踪的重新配置，这些跟踪是在运行时由跟踪源产生的。

使用调试器显示属性增强调试

解释如何用调试器显示属性增强调试功能。

[如何:在应用程序中跟踪代码](#)

解释如何使用允许检测应用程序的 Trace 类。

[如何:配置跟踪开关](#)

解释如何使用 .config 文件配置开关。

调试 Web 服务

[演练:调试 XML Web services](#)

提供调试 Web 服务的步骤。

调试 Windows 窗体

[演练:调试 Windows 窗体](#)

描述如何调试 Windows 窗体应用程序。

调试 SQL 应用程序

[演练:调试 SQL CLR 用户定义的表值函数](#)

显示如何调试 SQL/CLR 用户定义的表值函数 (UDF)。

[演练:调试 SQL CLR 触发器](#)

显示如何调试 SQL CLR 触发器。它使用 **AdventureWorks** 示例数据库（同 SQL Server 2005 一起安装的数据库之一）中的 Contact 表。此示例在 Contact 表中创建新的插入 CLR 触发器，然后单步执行它。

[演练:调试 SQL CLR 用户定义的类型](#)

显示如何调试 SQL/CLR 用户定义的类型。它在 **Adventureworks** 示例数据库中创建一个新的 SQL/CLR 类型。然后将此类型用于表定义、INSERT 语句、以及 SELECT 语句中。

[演练:调试 SQL CLR 用户定义的标量函数](#)

显示如何调试 SQL CLR 用户定义的函数 (UDF)。它在 **Adventureworks** 示例数据库中创建一个新的 SQL CLR 用户定义的函数。

[演练:调试 SQL CLR 用户定义的聚合](#)

显示如何调试 CLR SQL 用户定义的聚合。它在 **Adventureworks** 示例数据库中创建一个名为 Concatenate 的新的 CLR SQL 聚合函数。在 SQL 语句中调用此函数时，此函数将指定为其输入参数的列中的所有值连接在一起。

[T-SQL 数据库调试](#)

描述必需的安装步骤，并提供示例，演示如何调试多层应用程序。

[演练:调试 T-SQL 触发器](#)

讨论使用 **Adventureworks** 数据库的示例，此数据库包含带有 UPDATE 触发器的 Sales.Currency 表。此示例中包含更新表中某行内容的存储过程，因此会导致该触发器被激发。在该触发器中设置断点，并通过使用不同的参数执行该存储过程，可以跟踪触发器中的不同执行路径。

[演练:调试 T-SQL 用户定义的函数](#)

讨论一个使用 **Adventureworks** 数据库中现有用户定义的函数 (UDF) 的示例，此函数名为 ufnGetStock，它返回给定 ProductID 的存货项数。

[演练:调试 T-SQL 存储过程](#)

显示如何通过直接的数据库调试创建并调试 T-SQL 存储过程，即通过“服务器资源管理器”单步执行此存储过程。它还演示了不同的调试技术，例如设置断点、查看数据项等。

[其他资源](#)

访问该网站需要 Internet 连接。

[Visual Studio 2005 开发人员中心](#)

包含介绍如何开发和调试应用程序的大量文章和资源。该网站的内容定期更新。

[请参见](#)

[概念](#)

[如何实现 - C#](#)

数据访问 (如何实现 - C#)

此页链接到有关广泛使用的数据访问任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

常规

[如何: 安装示例数据库](#)

提供安装示例数据库(如 Northwind 示例数据库)、SQL Server 速成版 (SSE)、MSDE 或 Northwind 的 Access 版本的步骤。

[演练: 创建简单的数据应用程序](#)

提供创建数据应用程序的逐步骤的过程。

连接到 Visual Studio 中的数据

[“连接到 Visual Studio 中的数据”概述](#)

提供有关将应用程序连接到来自多个不同源(如数据库、Web 服务和对象)的数据的信息。

[演练: 连接到数据库中的数据](#)

提供使用“数据资源配置向导”将应用程序连接到 Visual Studio 中的数据的过程。

[演练: 连接到 Web 服务中的数据](#)

提供使用“数据资源配置向导”将应用程序连接到 Web 服务中的数据的过程。

[演练: 连接到 Access 数据库中的数据](#)

提供使用“数据资源配置向导”将应用程序连接到 Access 数据库中的数据的过程。

创建和设计类型化数据集

[如何: 创建类型化数据集](#)

解释如何使用“数据资源配置向导”或“数据集设计器”创建类型化数据集。

[演练: 使用数据集设计器创建数据集](#)

提供使用“数据集设计器”创建数据集的过程。

[演练: 在数据集设计器中创建数据表](#)

提供使用“数据集设计器”创建数据表的过程。

[演练: 创建数据表之间的关系](#)

解释如何使用“数据集设计器”并创建两个数据表之间的关系来创建两个不带 TableAdapter 的数据表。

TableAdapter

[TableAdapter 概述](#)

提供 TableAdapter 的概述, TableAdapter 用于提供应用程序与数据库之间的通信。

[演练: 创建带有多个查询的 TableAdapter](#)

提供使用“数据资源配置向导”在数据集中创建 TableAdapter 的过程。该演练将引导您完成以下过程:使用“数据集设计器”中的“TableAdapter 查询配置向导”在 TableAdapter 中创建其他查询。

填充数据集和执行查询

[“填充数据集和查询数据”概述](#)

解释如何使用 TableAdapter 或命令对象针对数据源执行 SQL 语句或存储过程。

[演练: 使用数据填充数据集](#)

演示如何创建带有一个数据表的数据集, 并用来自 Northwind 示例数据库中 Customers 表的数据填充该数据集。

[演练: 将 XML 数据读取到数据集](#)

演示如何创建要将 XML 数据加载到数据集中的 Windows 应用程序。

在 Windows 窗体上显示数据

[“显示数据”概述](#)

提供在创建数据绑定 Windows 应用程序时所涉及的任务、对象和对话框的摘要。

[演练:在 Windows 应用程序中的窗体上显示数据](#)

提供创建可将单个表中的数据显示在多个单独的控件中的简单窗体的过程。

[演练:在 Windows 应用程序中的窗体上显示相关数据](#)

提供处理来自多个表(并且通常来自相关表)的数据的过程。

[演练:在 Windows 应用程序中创建一个用于搜索数据的窗体](#)

显示如何创建返回特定城市的客户的查询、以及如何修改用户界面，以使用户能够输入城市名称然后按下按钮即可执行查询。

[演练:创建查找表](#)

提供根据另一个表中外键字段的值，显示来自一个表的信息的过程。

[数据绑定](#)

[演练:创建支持简单数据绑定的用户控件](#)

显示如何创建可实现 DefaultBindingPropertyAttribute 的控件。此控件可包含一个可以绑定到数据的属性；与 TextBox 或 CheckBox 相似。

[演练:创建支持复杂数据绑定的用户控件](#)

显示如何创建可实现 ComplexBindingPropertiesAttribute 的控件。此控件包含可以绑定到数据的 DataSource 和 DataMember 属性；与 DataGridView 或 ListBox 相似。

[演练:创建支持查找数据绑定的用户控件](#)

显示如何创建可实现 LookupBindingPropertiesAttribute 的控件。此控件包含三个可以绑定到数据的属性；与 ComboBox 相似。

[Visual Studio 中的对象绑定](#)

解释用于处理作为应用程序中的数据源的自定义对象(与数据集和 Web 服务相对)的设计时工具。

[编辑数据集中的数据\(数据表\)](#)

[编辑数据集中的数据概述](#)

提供一个表，其中包含到与编辑和查询数据集中的数据相关联的常规任务的链接。

[验证数据](#)

[数据验证概述](#)

提供验证数据的概述，验证数据是确认输入到数据对象中的值是否符合数据集架构中的约束以及为应用程序建立的规则的过程。

[演练:向数据集添加验证](#)

解释如何使用 ColumnChanging 事件验证输入到记录中的值是否可接受。

[保存数据](#)

[保存数据概述](#)

解释如何将向原始数据源写入信息的过程与修改数据集中数据的过程分开。

[ADO.NET 中的并发控制](#)

解释并发控制的常见方法，以及处理并发错误的特定 ADO.NET 功能。

[演练:用 TableAdapter DBDirect 方法保存数据](#)

提供有关使用 TableAdapter 的 DbDirect 方法针对数据库直接执行 SQL 语句的详细说明。

[演练:处理并发异常](#)

包含创建 Windows 应用程序的过程，该应用程序阐释了如何捕捉 DBConcurrency 异常、查找造成错误的行以及处理错误的一个策略。

[数据资源](#)

[数据用户界面元素](#)

包含有关在应用程序中设计数据访问时使用的所有对话框和向导的信息。

ADO.NET 数据适配器

提供有关 ADO.NET 数据适配器对象以及如何在 Visual Studio 中使用它们的信息。

在托管代码中创建 SQL Server 2005 对象

[SQL Server 项目](#)

解释除使用 Transact-SQL 编程语言外，如何使用 .NET 语言创建数据库对象（如存储过程和触发器），以及检索和更新 Microsoft SQL Server 2005 数据库中的数据。

[演练: 使用托管代码创建存储过程](#)

提供以下操作的分步说明：

在托管代码中创建存储过程。

将存储过程部署到 SQL Server 2005 数据库。

创建脚本以对数据库中的存储过程进行测试。

查询数据库中的数据，以确认存储过程是否正常执行。

其他资源

访问以下网站需要 Internet 连接。

[Visual Studio 2005 开发人员中心](#)

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

[Visual C# 开发人员中心](#)

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

[Microsoft .NET Framework 开发人员中心](#)

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

数据访问和存储开发人员中心

包含介绍如何在应用程序中使用 Microsoft 数据访问技术的大量文章和资源。

[SQL Server 开发人员中心](#)

包含有关使用 SQL Server 的大量文章和资源。

请参见

概念

[如何实现 - C#](#)

设计类(如何实现 - C#)

此页链接到广泛使用的 C# 类设计器任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

类设计器

[如何:在类关系图上创建类型](#)

描述如何创建新类型，例如类、枚举、接口、结构和委托。

[创建和配置类型成员](#)

描述如何使用类型成员。

[如何:从泛型类型继承](#)

解释如何建立类与泛型类之间的继承关系。

[如何:定义类型之间的继承关系](#)

解释如何使用类设计器定义当前显示在类关系图中的两个类型之间的继承关系。

[如何:定义类型之间的关联](#)

解释如何定义关联。类设计器中的关联行显示了关系图中各类之间的相互关系。

[如何:从类关系图中删除类型形状和关联的代码](#)

描述如何从类关系图中移除形状，或同时移除形状和代码。

[如何:对类型或类型成员应用自定义属性](#)

解释如何将自定义属性应用于类型或类型成员。

使用类和其他类型

[如何:查看类型之间的继承关系](#)

演示如何显示选定类型的基类型，假定该类型和其基类型之间存在继承关系。

[如何:查看派生类型](#)

显示由选定类型派生的类型。假定该类型和其基类或接口之间存在继承关系。

[如何:从类关系图中移除类型形状](#)

包含如何从关系图中移除形状的演示过程。

[如何:查看类型形状中的隔离舱](#)

包含如何显示或隐藏隔离舱的演示过程。

[如何:查看类型详细信息](#)

包含如何显示类型详细信息的演示过程。

[如何:在成员表示法与关联表示法之间转换](#)

包含如何在成员表示法和关联之间进行转换的演示过程。

[如何:查看类型成员](#)

包含如何显示或隐藏类型内部成员的演示过程。

[如何:向项目添加类关系图](#)

包含如何向项目中添加类关系图的演示过程。

[如何:查看现有类型](#)

包含如何在设计图面上使现有类型可视化的演示过程。

[如何:向项目添加类关系图](#)

包含如何向项目中添加类关系图的演示过程。

[了解他人编写的代码](#)

解释如何将 Visual Studio 类设计器作为工具，帮助自己理解他人编写的类和类型。该工具可显示代码的图形表示方式。可以自定义此视图以适合您的首选项。

[如何:对类型成员进行分组](#)

包含如何按类型、访问修饰符将成员分组，或如何按字母顺序将成员排序的演示过程。

[如何:向类关系图添加注释](#)

包含如何使用注释形状对类关系图进行批注的演示过程。

[自定义类关系图](#)

包含如何更改类关系图显示有关项目信息的方式的演示过程。

[如何:将类关系图元素复制到 Microsoft Office 文档](#)

包含如何将一个、多个或所有形状从类关系图复制到其他文档中的演示过程。

[如何:打印类关系图](#)

包含如何使用 Visual Studio 的打印功能打印类关系图的演示过程。

[如何:重写类型成员](#)

包含如何使用类设计器使子类中的成员重写(提供新的实现)从基类继承的成员的演示过程。

[如何:重命名类型和类型成员](#)

包含如何使用类设计器、“类详细信息”窗口或“属性”窗口重命名类型或类型成员的演示过程。

[如何:将类型成员从一个类型移到另一个类型中](#)

包含如何将类型成员从一个类型移动到另一个类型的演示过程(在当前类关系图中两种类型皆可见的情况下)。

[如何:实现接口](#)

解释如何使用类设计器创建、实现和删除接口。

[如何:实现抽象类](#)

包含如何使用类设计器实现抽象类的演示过程。

[如何:提取到接口\(仅限 C#\)](#)

包含如何将一个或多个公共成员从类型提取到新接口中的演示过程。

[如何:重新排列参数\(仅限 C#\)](#)

包含如何按类设计器中显示的类型重新排列方法的参数的演示过程。

[创建和修改类型成员](#)

[如何:打开“类详细信息”窗口](#)

描述如何使用“类详细信息”窗口配置类型成员。

[“类详细信息”窗口元素](#)

描述“类详细信息”窗口中显示的行的不同方面。

[如何:创建成员](#)

解释如何使用以下任意工具创建成员:类设计器、“类详细信息”窗口工具栏或“类详细信息”窗口。

[如何:向方法添加参数](#)

解释如何使用“类详细信息”窗口将参数添加到方法中。

[如何:修改类型成员](#)

描述如何使用“类详细信息”窗口修改在类设计器中创建的类型成员。

[“类详细信息”窗口使用说明](#)

提供使用“类详细信息”窗口的提示。

[显示只读信息](#)

解释类设计器和“类详细信息”窗口如何显示项目的类型(和类型成员),或显示项目中引用的项目或程序集的类型(和类型成员)。

类库设计指南

如何:实现控件的设计器

描述如何实现 HelpLabel 扩展程序提供程序控件的设计器 (HelpLabelDesigner)。

如何:在设计模式下创建和配置组件

演示如何使用设计器服务创建和初始化自定义设计器中的组件。

如何:在 Windows 窗体中访问设计时支持

提供访问由 .NET Framework 提供的设计时支持的步骤。

如何:实现 HelpLabel 扩展程序提供程序

演示如何通过创建 HelpLabel 控件生成扩展程序提供程序。

如何:访问设计时服务

说明如何获得对一组丰富的 .NET Framework 服务的访问,以便将组件和控件集成到设计环境中。

如何:使用 DesignerSerializationVisibilityAttribute 序列化标准类型的集合

演示如何使用 DesignerSerializationVisibilityAttribute 类控制设计时集合被序列化的方式。

如何:在设计模式下执行控件的自定义初始化

演示如何在设计环境创建控件时初始化该控件。

如何:实现类型转换器

演示如何使用类型转换器在数据类型之间转换值,并通过提供文本到值的转换或待选值的下拉列表协助设计时的属性配置。

如何:实现用户界面类型编辑器

演示如何实现 Windows 窗体的自定义 UI 类型编辑器。

如何:在设计模式下扩展控件的外观和行为

演示如何创建自定义设计器,该设计器扩展用户界面 (UI) 用于设计自定义控件。

如何:创建利用设计时功能的 Windows 窗体控件

阐释如何创建自定义控件和关联的自定义设计器。生成此库后,便可以生成在窗体上运行的自定义 MarqueeControl 实现。

如何:向 Windows 窗体组件附加智能标记

说明如何向组件和自定义控件添加智能标记支持。

如何:在设计模式下调整组件的属性 (Attribute)、事件及属性 (Property)

演示如何创建调整组件的特性、事件和属性的自定义设计器。

其他资源

访问以下网站需要 Internet 连接。

Visual Studio 2005 开发人员中心

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

Visual C# 开发人员中心

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

Microsoft .NET Framework 开发人员中心

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

Microsoft 模式与实践开发人员中心

提供了特定于方案的建议做法,阐释如何在 Microsoft .NET 平台上,从体系结构的角度设计、开发、部署和操作健全可靠的应用程序。

请参见

概念

如何实现 - C#

安全性(如何实现 - C#)

此页面链接到广泛使用的安全和部署任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

常规

[Visual Studio 中的安全性](#)

提供安全编码技术的介绍。

[代码访问安全性基础知识](#)

提供代码访问安全性的概念，以编写针对公共语言运行库的有效应用程序。

[Microsoft 安全开发人员中心](#)

提供最新的安全问题，帮助您开发安全代码。

[新技术使您的 Web 服务更安全](#)

解释良好的安全性的重要性，以及对 Web 服务的影响。

[确定何时使用 Windows Installer 与 XCOPY](#)

检查和比较部署 Microsoft .NET 应用程序的两种方法：DOS XCOPY 命令和 Microsoft Windows Installer 技术。

[安全策略最佳实施策略](#)

解释基本管理概念，并描述管理代码访问安全策略时应使用的一些最佳做法。

[代码访问和权限集](#)

[如何：使用数据保护](#)

提供使用数据保护加密或解密内存中的数据、文件或流的步骤。

[如何：向安全策略添加自定义权限](#)

提供将自定义权限添加到安全策略中的过程。

[如何：对托管执行启用 Internet Explorer 安全设置](#)

提供启用 Internet Explorer 安全设置的过程。

[如何：通过使用 RequestMinimum 标志请求最小权限](#)

提供使用 RequestMinimum 标志请求 FileIOPermission 的示例。

[如何：创建 GenericPrincipal 和 GenericIdentity 对象](#)

提供如何使用 GenericIdentity 类与 GenericPrincipal 类一起创建独立于 Windows NT 或 Windows 2000 域而存在的授权方案的示例。

[如何：创建 WindowsPrincipal 对象](#)

提供创建 WindowsPrincipal 对象的两种方法，具体取决于代码是必须重复执行基于角色的验证还是必须只执行一次。

[如何：执行命令性安全检查](#)

提供使用命令性检查确保 GenericPrincipal 同 PrincipalPermission 对象相匹配的示例。

[如何：通过使用 RequestRefuse 标志拒绝授予权限](#)

提供使用 RequestRefuse 拒绝来自公共语言运行库安全系统的 FileIOPermission 的示例。

[如何：请求访问非托管代码的权限](#)

提供示例，显示如何请求权限以访问非托管代码。

[如何：请求命名权限集的权限](#)

提供示例，显示请求指定的权限集的权限的语法。

[如何：通过使用 RequestOptional 标志请求可选权限](#)

提供示例，使用 SecurityAction.RequestOptional 标志请求 FileIOPermission，这样可以间接拒绝其他所有权限。

如何:将非对称密钥存储在密钥容器中

演示如何创建非对称密钥，将其保存在密钥容器中，稍后检索此密钥，最后从容器中删除此密钥。

如何:使用 Caspol.exe 向安全策略添加程序集

解释如何将实现自定义安全对象的程序集添加到完全受信任的程序集列表。

如何:使用 Caspol.exe 查看代码组

解释如何使用代码访问安全策略工具 (Caspol.exe) 查看属于某个策略级别的代码组的简单列表，或包括代码组的名称和说明的列表。

如何:修改权限集中的权限

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 修改权限集中的权限。

如何:向权限集添加权限

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 向权限集中添加权限。

如何:使用 Caspol.exe 取消显示策略更改警告

解释如何使用 Caspol.exe 取消策略更改警告。

如何:更改代码组的成员条件

解释如何使用 Mscorcfg.msc 更改与代码组相关的成员条件。

如何:使用 Caspol.exe 查看代码组和权限集

解释如何使用代码访问安全策略工具 (Caspol.exe) 列出程序集所属的全部代码组。

如何:使用 Caspol.exe 管理非默认用户的安全策略

解释如何使用代码访问安全策略工具 (Caspol.exe) 管理非当前用户的用户策略。

如何:更改与现有代码组关联的权限集

解释如何使用 Mscorcfg.msc 更改权限集。

如何:使用 Caspol.exe 分析程序集权限的问题

解释如何使用代码访问安全策略工具 (Caspol.exe) 解决可能导致程序集出现的以下问题：不运行、访问受保护的资源或在不应运行时运行。

如何:使用 Caspol.exe 查看权限集

解释如何使用代码访问安全策略工具 (Caspol.exe) 列出属于所有策略级别或属于单个策略级别的权限集。

如何:使用 Caspol.exe 撤消策略更改

解释如何使用代码访问安全策略工具 (Caspol.exe)，在做出更改前恢复上一计算机、用户或企业策略。

如何:使用 XML 文件导入权限

提供一个示例，显示权限信息在 XML 文件中可能如何出现。

如何:使用 Caspol.exe 恢复默认安全策略设置

解释如何使用 Caspol.exe 返回到默认安全策略设置。

如何:使用 Caspol.exe 添加代码组

解释如何使用 Caspol.exe 添加代码组。

如何:重写 Caspol.exe 自保护机制

解释如何在必要情况下重写自保护机制。

如何:创建代码组

解释如何使用 Mscorcfg.msc 创建代码组。

如何:禁用并发垃圾回收

解释如何使用 <gcConcurrent> 元素指定运行库应如何运行垃圾回收。

如何: 使用 XML 文件导入代码组

提供一个示例，显示代码组及其关联的成员条件和权限集的信息在 XML 文件中可能如何出现。

如何: 创建发行者策略

提供一个示例，显示发行者策略文件，该文件将 myAssembly 的一个版本重定向到另一个版本。

如何: 使用 Caspol.exe 移除代码组

解释如何使用代码访问安全策略工具 (Caspol.exe) 从代码组层次结构中移除代码组。

如何: 在配置文件中创建一个信道模板

提供一个示例，说明如何在配置文件中创建信道模板。

如何: 使用 Caspol.exe 更改权限集

解释如何使用代码访问安全策略工具 (Caspol.exe) 将原始权限集替换为在 XML 文件中指定的新权限集。

如何: 移除权限集

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 移除处于特定级别的权限集。

如何: 创建权限集

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 创建特定级别的权限集，并将它与新的或现有的代码组关联。

如何: 使代码组成为独占的或最终级别的

解释如何使用 Mscorcfg.msc 使新代码组成为独占的或最终级别的。

如何: 向策略程序集列表添加程序集

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc)，将程序集添加到完全受信任的程序集列表。

如何: 使用 XML 文件导入权限集

提供一个示例，显示权限集和 XML 文件内的权限。

如何: 使用 DEVPATH 查找程序集

提供一个示例，说明如何使运行库在由 DEVPATH 环境变量所指定的目录中搜索程序集。

如何: 为宿主应用程序域注册服务器激活的对象和客户端激活的对象

提供一个示例，说明如何为宿主应用程序域注册服务器端激活的对象和客户端激活的对象。

如何: 使用 Caspol.exe 查看安全策略

解释如何使用代码访问安全策略工具 (Caspol.exe) 查看安全策略代码组层次结构，以及所有策略级别或单个策略级别的已知权限集的列表。

如何: 使用 Caspol.exe 添加权限集

解释如何使用代码访问安全策略工具 (Caspol.exe) 向代码组中添加权限集。

如何: 使用 Caspol.exe 更改代码组

解释如何使用代码访问安全策略工具 (Caspol.exe) 的 **-chggrou**p 选项，更改代码组的名称、成员条件、权限集、标志或说明。

如何: 配置信道

提供一个示例，说明如何用不同于“http”的名称生成一个 HttpChannel，并将其用于服务器应用程序。

如何: 使用 Caspol.exe 打开和关闭安全性

解释如何使用代码访问安全策略工具 (Caspol.exe) 打开和关闭安全性。

如何: 从权限集中移除权限

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 从权限集中移除权限。

如何: 使用 .NET Framework 配置工具 (Mscorcfg.msc) 执行常见的安全策略任务

解释如何使用 .NET Framework 配置工具 (Mscorcfg.msc) 配置安全策略，以满足您的需要。

其他资源

[Microsoft 安全开发人员中心](#)

包含介绍如何开发安全的应用程序的大量文章和资源。

[Visual Studio 2005 开发人员中心](#)

包含介绍如何使用 Visual Studio 2005 开发应用程序的大量文章和资源。该网站的内容定期更新。

[Visual C# 开发人员中心](#)

包含介绍如何开发 C# 应用程序的大量文章和资源。该网站的内容定期更新。

[Microsoft .NET Framework 开发人员中心](#)

包含介绍如何开发和调试 .NET Framework 应用程序的大量文章和资源。该网站的内容定期更新。

[请参见](#)

[概念](#)

[如何实现 - C#](#)

Office 编程 (如何实现 - C#)

此页面链接到广泛使用的 Office 编程任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

常规

[如何:升级文档级项目](#)

解释为完成到 Microsoft Visual Studio 2005 Tools for Office 的升级所需手动执行的步骤。

[使用 Excel 的演练](#)

演示三种基本类型的任务：实现 Microsoft Office Excel 2003 自动化、执行数据分析和使用控件。

[使用 Word 的演练](#)

演示如何使用 Microsoft Office 2003 的工具实现 Microsoft Office Word 2003 项目的自动化。

[如何:使用主互操作程序集使 Office 应用程序自动化](#)

提供在现有项目中使用 Visual Basic 或 C# 调用非托管代码需执行的步骤。

[如何:向 Office 文档添加控件](#)

解释如何在设计时或运行时向 Office 文档中添加控件。

Word 和 Excel 应用程序

[如何:以编程方式运行 Excel 计算](#)

解释如何以编程方式对部分或全部应用程序运行计算。

[如何:以编程方式创建 Office 菜单](#)

提供示例，在 Microsoft Office Excel 2003 菜单栏上创建称为“新菜单”的菜单。

[如何:以编程方式创建 Office 工具栏](#)

提供示例，在 Microsoft Office Word 2003 中创建称为“测试”的工具栏。它显示在接近文档中部的位置，包含两个按钮。单击按钮后，出现消息框。

[如何:处理 Office 项目中的错误](#)

解释如何将调试器设置为在发生公共语言运行库异常时中断。

请参见

概念

[如何实现 - C#](#)

智能设备

此页面链接到广泛使用的智能设备编程任务的“帮助”。若要查看“帮助”中涵盖的其他类别的常用任务，请参见[如何实现 - C#](#)。

智能设备入门

[智能设备项目中的新增功能](#)

提供 Visual Studio 2005 中可用的新功能或扩展功能。

[设备功能和所需的开发工具](#)

提供表格，其中包含智能设备硬件、硬件功能和开发工具的变化的快照。

[设备项目的远程工具](#)

提供在嵌入式 Visual C++ 4.0 中可用的、随 Visual Studio 2005 一起提供、协助开发和调试设备应用程序的远程工具列表。

[如何:优化智能设备开发帮助](#)

显示如何确定已安装的帮助、如何添加移动和嵌入式开发帮助、以及如何筛选帮助。

[如何:在 Visual Studio 中启动设备仿真程序](#)

显示如何使用“连接到设备”对话框或设备仿真器管理器启动设备仿真程序。

[更新由以前的工具创建的项目](#)

提供 Visual Studio 2005 开发环境的改善。

[选择开发语言](#)

开发应用程序、控件或库以用于在智能设备上进行部署时，提供编程语言选项。

[演练:创建用于设备的 Windows 窗体应用程序](#)

说明桌面和设备编程之间的主要差异；即，必须将一个设备指定为目标。在此演练中，该设备为 Pocket PC 2003 的内置仿真程序。

[使用 Visual Basic 和 C# 进行智能设备编程](#)

[使用 .NET Compact Framework 进行设备编程](#)

提供使用 Visual Basic 或 C# 语言及 .NET Compact Framework 开发智能设备应用程序的信息。

[设备项目的 .NET Compact Framework 引用](#)

提供 .NET Compact Framework(作为 .NET Framework 类库的子集)的信息。

[如何:使用 Visual Basic 或 Visual C# 创建设备应用程序](#)

介绍如何创建设备应用程序，以及该创建过程与创建桌面应用程序的过程有何不同。

[如何:跨平台共享源代码\(设备\)](#)

介绍如何使用编译器常数在多个不同平台间共享相同的源代码。

[如何:在设备项目中更改平台](#)

介绍如何在同一项目中在不同平台间来回切换。

[如何:将项目升级到更高版本的 .NET Compact Framework](#)

介绍在安装了更新版本的平台的情况下，如何升级现有项目的平台。

[在设备项目中管理代码段](#)

介绍如何使用专属于设备项目的代码段。

[如何:在设备项目中验证代码的平台支持](#)

介绍如何确保目标平台支持您的代码。

[如何:处理 HardwareButton 事件\(设备\)](#)

介绍如何在 Pocket PC 上重写应用程序键。

如何:更改窗体的方向和分辨率(设备)

介绍在默认方向或分辨率不正确或缺失的情况下, 如何更改方向和分辨率。

如何:更改默认设备(托管项目)

介绍如何在项目开发过程中更改目标设备。

如何:优化智能设备开发帮助

介绍如何使用智能设备帮助筛选器仅显示设备应用程序开发所支持的那些 .NET Framework 元素。

设备连接

如何:从虚拟 PC 会话连接到设备仿真程序

描述在缺少 TCP/IP 的情况下的连接技术。

如何:访问 Smartphone 仿真程序文件系统

描述如何访问 Smartphone 仿真程序的文件系统, 该仿真程序没有自己的文件查看器。

如何:使用蓝牙连接

描述如何使用蓝牙进行连接。

如何:使用 IR 进行连接

描述如何使用红外线进行连接。

如何:在不使用 ActiveSync 的情况下连接到 Windows CE 设备

描述在无法使用 ActiveSync 服务时连接到设备所需的步骤。

如何:从仿真程序访问开发计算机文件

描述如何使用共享文件夹从仿真程序访问开发计算机文件。

如何:设置连接选项(设备)

描述在何处可以查找用于设置连接选项的通用对话框。

调试智能设备

设备和桌面调试器之间的差异

描述设备和桌面调试器之间的差异。

如何:附加到托管设备进程

描述如何附加到托管设备进程。

如何:更改设备注册表设置

描述如何使用远程注册表编辑器更改设备的注册表设置。

演练:调试同时包含托管代码和本机代码的解决方案

介绍如何调试这些混合解决方案。

托管设备项目中的数据

如何:生成 SqlCeResultSet 代码(设备)

介绍如何生成 ResultSet 而不是生成 DataSet。

如何:更改设计时连接字符串(设备)

介绍如何在设计时更改 Visual Studio 用来连接到 SQL Server Mobile 数据库的字符串。

如何:更改运行时连接字符串(设备)

介绍如何在运行时更改应用程序用来连接到 SQL Server Mobile 数据库的字符串。

如何:添加导航按钮(设备)

介绍 **DataNavigator** 类的替代选择, .NET Compact Framework 不支持该类。

如何:将数据更改持久地保存到数据库中(设备)

介绍如何将数据集中的更改应用回数据库。

[如何: 创建数据库\(设备\)](#)

介绍如何使用 Visual Studio 环境创建 SQL Mobile 数据库(在项目内部或外部)。

[如何: 向设备项目添加数据库](#)

介绍如何将 SQL Mobile 数据库(可在“服务器资源管理器”中找到)添加为 Visual Basic 或 Visual C# 项目的数据源。

[如何: 添加 SQL Server 数据库作为数据源\(设备\)](#)

介绍如何将 SQL Server 数据库添加为 Visual Basic 或 Visual C# 项目的数据源。

[如何: 添加业务对象作为数据源\(设备\)](#)

介绍如何将业务对象添加为 Visual Basic 或 Visual C# 项目的数据源。

[如何: 添加 Web 服务作为数据源\(设备\)](#)

介绍如何将 Web 服务添加为 Visual Basic 或 Visual C# 项目的数据源。

[如何: 管理数据库中的表\(设备\)](#)

介绍如何添加和移除表, 以及如何编辑现有表的架构。

[如何: 管理数据库中的列\(设备\)](#)

介绍如何添加和移除列以及如何编辑它们的属性。

[如何: 管理数据库中的索引\(设备\)](#)

介绍如何添加和移除索引, 以及如何更改它们的排序顺序属性。

[如何: 管理数据库密码\(设备\)](#)

介绍如何为新的 SQL Mobile 数据库设置密码, 以及如何更改现有数据库的密码。

[如何: 缩小和修复数据库\(设备\)](#)

介绍如何压缩和修复 SQL Mobile 数据库。

[如何: 创建参数化查询\(设备\)](#)

介绍如何创建参数化查询。

[演练: 参数化查询应用程序](#)

为包括构建参数化查询的端到端项目提供分步指导。

[如何: 创建主/从应用程序\(设备\)](#)

介绍如何实现主/从关系。

[演练: 数据库主/从应用程序](#)

为包括创建和运行主/从应用程序的端到端项目提供分步指导。

[如何: 预览数据库中的数据\(设备\)](#)

介绍几种查看数据库中的数据的方式。

[如何: 为数据应用程序生成摘要视图和编辑视图\(设备\)](#)

介绍如何使用数据窗体查看和编辑数据网格中的单行数据。

[打包和部署设备解决方案](#)

[演练: 打包智能设备解决方案以便进行部署](#)

提供对应用程序及其资源进行打包的分步指导。

[设备项目中的安全性](#)

[如何: 在设备项目中导入和应用证书](#)

说明如何有效使用“选择证书”对话框对设备项目进行签名。

[如何: 将 Signtool.exe 作为生成后事件启动\(设备\)](#)

说明如何在生成后事件已更改原始二进制文件后对项目进行签名。

[如何:向设备查询其安全模型](#)

说明如何确定设备证书存储区中已安装了哪些证书。

[如何:对 Visual Basic 或 Visual C# 应用程序进行签名\(设备\)](#)

列出对针对 .NET Compact Framework 编写的应用程序进行签名的步骤。

[如何:对 Visual Basic 或 Visual C# 程序集进行签名\(设备\)](#)

列出对项目程序集进行签名的步骤。

[如何:对 CAB 文件进行签名\(设备\)](#)

列出对设备 CAB 项目进行签名的步骤。

[如何:在 Visual Basic 或 Visual C# 项目中提供设备](#)

列出向托管项目的设备存储区中添加数字证书的步骤。

[如何:为设备提供安全模型](#)

说明如何使用 RapiConfig.exe 配置具有安全模型的设备。

[请参见](#)

[概念](#)

[如何实现 - C#](#)

部署 (如何实现 - C#)

此页面链接到有关广泛使用的部署任务的帮助。若要查看“帮助”中涵盖的其他类别的常用任务，请参见如何实现 - C#。

ClickOnce

[如何:发布 ClickOnce 应用程序](#)

演示如何通过将 ClickOnce 应用程序发布到 Web 服务器、文件共享或可移动媒体来供用户使用。

[如何:指定发布位置](#)

演示如何指定放置应用程序文件和清单的位置。

[如何:指定安装 URL](#)

演示如何使用“安装 URL”属性指定用户用来下载应用程序的 Web 服务器。

[如何:指定支持 URL](#)

演示如何使用“支持 URL”属性标识用户可以从中获取有关应用程序的信息的网页或文件共享。

[如何:指定 ClickOnce 安装模式](#)

演示如何设置“安装模式”以指定应用程序是脱机使用还是联机使用。

[如何:为 CD 安装启用自动启动](#)

演示如何启用“自动启动”以便在插入媒体时自动启动 ClickOnce 应用程序。

[如何:设置 ClickOnce 发布版本](#)

演示如何设置“发布版本”属性以确定是否将要发布的应用程序视为更新。

[如何:自动递增 ClickOnce 发布版本](#)

演示如何更改“发布版本”属性以使应用程序作为更新发布。

[如何:指定通过 ClickOnce 发布的文件](#)

演示如何排除文件，将文件标记为数据文件或系统必备，并创建用于有条件安装的文件组。

[如何:与 ClickOnce 应用程序一起安装系统必备组件](#)

演示如何指定要与应用程序一起打包的一组系统必备组件。

[如何:管理 ClickOnce 应用程序的更新](#)

演示如何指定执行更新检查的时间和方式，是否强制更新，以及应用程序应在何处进行更新检查。

[如何:将数据文件包括到 ClickOnce 应用程序中](#)

提供演示如何将任意类型的数据文件添加到 ClickOnce 应用程序中的步骤。

[如何:使用 Systems Management Server 部署 .NET Framework](#)

提供必须在运行 Systems Management Server 的服务器上执行的任务。

[如何:为 ClickOnce 应用程序向客户端计算机添加一个受信任的发行者](#)

演示如何使用命令行工具 CertMgr.exe 将发行者的证书添加到客户端计算机上“受信任的发行者”存储区。

[如何:指定部署更新的其他位置](#)

演示如何在部署清单中指定更新的备用位置，使应用程序在初始安装后可从网站自行更新。

[如何:使用 Active Directory 部署 .NET Framework](#)

提供使用 Active Directory 部署 .NET Framework 的过程。

[如何:在 ClickOnce 应用程序中检索查询字符串信息](#)

提供演示如何使用 ClickOnce 应用程序获取查询字符串信息的过程。此外，它还演示第一次启动 ClickOnce 应用程序时，该应用程序如何使用一小段代码读取这些值。

[如何:启用 ClickOnce 安全设置](#)

演示如何在开发过程中临时禁用安全设置。

如何:为 ClickOnce 应用程序设置安全区域

演示如何设置安全区域以填充应用程序表所要求的权限。

如何:设置 ClickOnce 应用程序的自定义权限

演示如何将应用程序限制为正常运行所需的特定权限。

如何:确定 ClickOnce 应用程序的权限

演示如何通过运行“权限计算器”工具分析应用程序以确定它所要求的权限。

如何:使用受限权限对 ClickOnce 应用程序进行调试

演示如何使用与最终用户相同的权限调试应用程序。

Windows Installer

Windows Installer 部署

提供指向一些文章的链接，这些文章介绍如何使用 Windows Installer 部署创建将分发给用户的安装程序包。

演练:部署基于 Windows 的应用程序

演示为一个用来启动“记事本”的 Windows 应用程序创建安装程序的过程。

演练:使用合并模块安装共享组件

演示如何合并模块，确保将共享组件一起打包和分发，从而实现一致的部署。

演练:创建自定义操作

演示如何创建一个 DLL 自定义操作，以便在安装结束时将用户直接链接到某个网页。

演练:安装时使用自定义操作显示消息

演示如何使用自定义操作接受用户输入，并将用户输入传递给安装期间出现的消息框。

演练:安装时使用自定义操作对程序集进行预编译

演示如何将 DLL 的路径名传递给 CustomActionData 属性，以便在安装过程中将程序集预编译为本机代码。

演练:安装时使用自定义操作创建数据库

演示如何在安装期间使用自定义操作和 CustomActionData 属性创建数据库和数据库表。

演练:安装时重定向应用程序以面向另一个 XML Web services

说明如何使用 URL Behavior 属性、Installer 类和 Web 安装项目，创建可重定向以面向不同的 XML Web services 的 Web 应用程序。

如何:在 Windows Installer 部署中安装系统必备组件

演示如何在安装期间自动检测组件是否存在，并安装一组预定的系统必备，此过程称作“引导”。

如何:创建或添加部署项目

演示如何指定解决方案在开发过程中和开发完成后的部署位置和部署方式。

如何:创建或添加安装项目

演示如何创建 Windows Installer (.msi) 文件，这些文件用于分发应用程序以将它们安装到其他计算机或 Web 服务器上。

如何:创建或添加合并模块项目

演示如何创建一个合并模块项目，以将在多个应用程序之间进行共享的文件或组件打包在一起。

如何:创建或添加 Cab 项目

演示如何创建一个 CAB 项目，以创建能够用来将组件下载到 Web 浏览器的压缩文件 (.cab)。

如何:设置部署项目属性

演示如何使用“部署属性”对话框设置与配置相关的属性。

如何:向部署项目中添加项

演示如何指定需要在安装程序中包括的内容，以及在目标计算机上的安装位置。

[如何:向部署项目中添加合并模块](#)

演示如何使用合并模块(.msm 文件)在多个部署项目之间共享组件。

[如何:添加和移除图标](#)

演示在安装期间如何在目标计算机上安装图标并将其与应用程序关联。

[如何:从部署项目中排除项](#)

演示如何从部署项目中排除一个或多个文件。

[如何:基于操作系统版本设置条件安装](#)

演示如何设置 Condition 属性，将条件逻辑添加到安装程序(例如，根据不同的操作系统版本安装不同的文件或设置不同的注册表值)。

请参见

概念

[如何实现 - C#](#)

[部署 C# 应用程序](#)

使用 Visual C# IDE

本节将介绍 Visual C# 集成开发环境 (IDE)，并描述如何在开发周期的所有阶段（从设置项目到将已完成的应用程序分发到最终用户）进行使用。

本节内容

[IDE 简介 \(Visual C#\)](#)

提供组成 Visual C# 集成开发环境的编辑器和窗口指南。

[创建项目 \(Visual C#\)](#)

描述如何为将要创建的应用程序类型设置正确的项目。

[设计用户界面 \(Visual C#\)](#)

解释如何使用设计器向应用程序中添加控件。

[编辑代码 \(Visual C#\)](#)

描述如何使用代码编辑器输入源代码，以及如何使用各种工具，如 IntelliSense、代码段和重构。

[导航和搜索 \(Visual C#\)](#)

描述如何快速在项目文件中移动和查找内容。

[生成和调试 \(Visual C#\)](#)

描述如何利用项目文件创建可执行程序集，以及如何在 Visual Studio 调试器中运行该程序集。

[建模和分析代码 \(Visual C#\)](#)

描述如何用类设计器查看类关系，如何用对象测试工作台测试对象，以及如何运行代码分析工具。

[添加和编辑资源 \(Visual C#\)](#)

描述如何将文件（例如图标、字符串、表和其他类型的数据）添加到项目中。

[获得帮助 \(Visual C#\)](#)

解释如何查找您所需要的文档。

[部署 C# 应用程序](#)

描述如何将应用程序分发给最终用户。

[Visual C# 代码编辑器功能](#)

提供有关 Visual C# 独有的 IDE 功能的完整参考材料，包括代码段和重构。

[Visual C# IDE 设置](#)

描述如何更改默认 Visual C# 设置。

[Visual C# 键盘快捷键](#)

解释如何加快使用 Visual C# 的速度。

请参见

其他资源

[Visual C#](#)

[Visual C# 入门](#)

[C# 参考](#)

[Visual Studio 集成开发环境](#)

IDE 简介 (Visual C#)

Visual C# 集成开发环境 (IDE) 是一种通过常用用户界面公开的开发工具的集合。有些工具是与其他 Visual Studio 语言共享的，还有一些工具（如 C# 编译器）是 Visual C# 特有的。本节中的文档提供如何在使用 IDE 时针对开发过程的各个阶段使用最重要的 Visual C# 工具的概述。

注意

如果您正在开发 ASP.NET 2.0 Web 应用程序，将使用 Visual Web Developer IDE，它是 Visual Studio 2005 的一个完全集成部分。但是，如果您的代码隐藏页是用 Visual C# 编写的，则会使用 Visual Web Developer 中的 Visual C# 代码编辑器。因此，本节中的某些主题（如[设计用户界面 \(Visual C#\)](#)）可能不完全适用于 Web 应用程序。

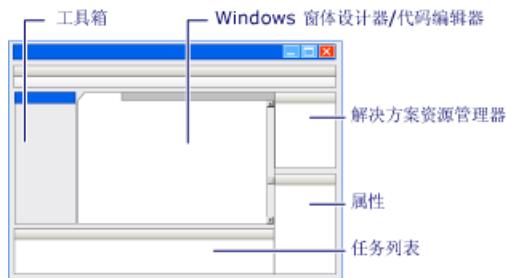
Visual C# 工具

以下是 Visual C# 中最重要的工具和窗口。大多数工具的窗口可从“视图”菜单打开。

- 代码编辑器，用于编写源代码。
- C# 编译器，用于将 C# 源代码转换为可执行程序。
- Visual Studio 调试器，用于对程序进行测试。
- “工具箱”和“设计器”，用于使用鼠标迅速开发用户界面。
- “解决方案资源管理器”，用于查看和管理项目文件和设置。
- “项目设计器”，用于配置编译器选项、部署路径、资源及更多其他内容。
- “类视图”，用于根据类型（而不是文件）在源代码中导航。
- “属性窗口”，用于配置用户界面中控件的属性和事件。
- 对象浏览器，用于查看动态链接库（包括 .NET Framework 程序集和 COM 对象）中可用的方法和类。
- 文档资源管理器，用于在本地计算机和 Internet 上浏览和搜索产品文档。

IDE 如何提供工具

可以通过 IDE 中的窗口、菜单、属性页和向导与这些工具进行交互。基本 IDE 的外观如下所示：



您可以通过按 Ctrl + Tab，快速访问所有打开的工具窗口或文件。有关更多信息，请参见[导航和搜索 \(Visual C#\)](#)。

编辑器窗口和 Windows 窗体设计器窗口

代码编辑器和 Windows 窗体设计器都使用该大型主窗口。通过按 F7，或者单击“视图”菜单上的“代码”或“设计器”，可在“代码”视图和“设计”视图之间进行切换。在“设计”视图中，可以将控件从“工具箱”（通过单击左边距上的“工具箱”选项卡即可看到）拖动到窗口。有关代码编辑器的更多信息，请参见[编辑代码 \(Visual C#\)](#)。有关 Windows 窗体设计器的更多信息，请参见[Windows 窗体设计器](#)。

右下方的“属性”窗口仅在“设计”视图中才会被生成。该窗口使您可以设置属性并挂接用户界面控件（如按钮、文本框等）的事件。如果将此窗口设置为“自动隐藏”，则只要切换至“代码”视图此窗口就会折叠进右边距。有关“属性”窗口和设计器的更多信息，请参见[设计用户界面 \(Visual C#\)](#)。

解决方案资源管理器和项目设计器

右上方的窗口是“解决方案资源管理器”，该窗口以分层树视图的方式显示项目中的所有文件。如果使用“项目”菜单将新文件添加到项目，将看到这些文件反映在“解决方案资源管理器”中。除文件外，“解决方案资源管理器”还显示项目设置，以及对应用程序所需的外部库的引用。

可以通过右击“解决方案资源管理器”中的“属性”节点，然后单击“打开”访问“项目设计器”属性页。使用这些页可以修改生成选项、安全要求、部署详细信息以及许多其他项目属性。有关“解决方案资源管理器”和“项目设计器”的更多信息，请参见[创建项目 \(Visual C#\)](#)。

编译器、调试器和错误列表窗口

C# 编译器没有窗口，因为它不是交互式工具，但可以在“项目设计器”中设置编译器选项。如果单击“生成”菜单上的“生成”，IDE 将调用 C# 编译器。如果生成成功，则状态窗格将显示“生成成功”消息。如果存在生成错误，将在编辑器/设计器窗口的下方出现带有错误列表的“错误列表”窗口。双击某个错误可以转到源代码中相应的问题行。按 F1 可以查看针对突出显示的错误的帮助文档。

调试器具有多个不同的窗口，这些窗口随着应用程序的运行显示变量的值和类型信息。在调试器中调试时，可以使用“代码编辑器”窗口指定在某一行暂停执行，以及每次一行单步执行代码。有关更多信息，请参见[生成和调试 \(Visual C#\)](#)。

自定义 IDE

Visual C# 中的所有窗口都可以变为可停靠或浮动、隐藏或可见，也可以移动到新位置。若要更改窗口的行为，请单击向下箭头或标题栏上的图钉图标，然后从可用选项中进行选择。若要将停靠窗口移动到新的停靠位置，请拖动标题栏，直至出现该窗口的滴管图标。按住鼠标左键的同时，将鼠标指针移至新位置该图标的上方。将指针放在左侧、右侧、顶端或底端图标上方，可使该窗口停靠在指定一侧。将指针放在中间图标的上方，可使该窗口成为选项卡式窗口。在您放置指针时，将出现一个蓝色半透明的矩形，它指示该窗口将停靠在新位置的什么地方。



通过单击“工具”菜单上的“选项”，可以对 IDE 的许多其他方面进行自定义。有关更多信息，请参见[“选项”对话框 \(Visual Studio\)](#)。

请参见

概念

[设计用户界面 \(Visual C#\)](#)

[创建项目 \(Visual C#\)](#)

[编辑代码 \(Visual C#\)](#)

[生成和调试 \(Visual C#\)](#)

其他资源

[Visual C#](#)

[使用 Visual C# IDE](#)

[Visual Studio 集成开发环境](#)

[Visual Web Developer](#)

[Visual Web Developer 用户界面元素](#)

创建项目 (Visual C#)

准备好开始编写代码后，第一步是设置项目。项目包含应用程序的所有原始资料，不仅包括源代码文件，还包括资源文件，如图标、对程序依赖的外部文件的引用，以及配置数据(如编译器设置)。生成项目时，Visual C# 调用 C# 编译器和其他内部工具，以使用项目中的文件创建可执行程序集。

创建新项目

可以通过单击“文件”菜单，指向“新建”，然后单击“项目”来创建新项目。

注意

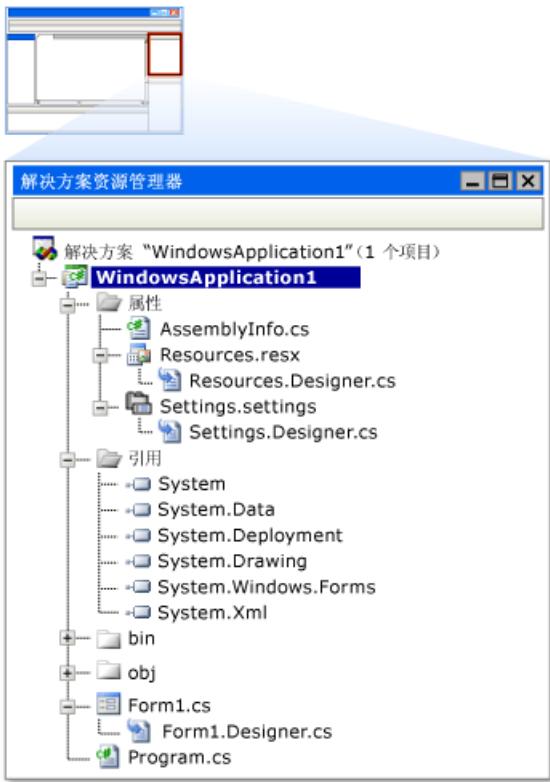
如果选择了“网站”而不是“项目”，则 [Visual Web Developer 集成开发环境 \(IDE\)](#) 将打开。此开发环境是 Visual Studio 中一种独特的独立环境，用于创建 ASP.NET Web 应用程序。Visual Web Developer IDE 不使用 Visual C# 代码编辑器编辑 C# 中的代码隐藏文件。如果您正在创建 Web 应用程序，应该主要使用 Visual Web Developer 文档，但有关 C# 编辑器的信息，还请参考[编辑代码 \(Visual C#\)](#)。

下图显示了“新建项目”对话框。可以看到默认情况下选择了左侧窗口中的“Visual C#”，而在右侧，可以从六种或更多项目模板中进行选择。如果展开左侧的“智能设备”或“其他项目类型”节点，可以看到在右侧显示不同的项目类型。



初学者工具包是另一种类型的项目模板。如果您安装了初学者工具包，将会看到该工具包在“新建项目”对话框中列出。有关更多信息，请参见[初学者工具包](#)。

选择项目模板并单击“确定”后，Visual Studio 将创建项目，您就可以开始编写代码了。项目文件、引用、设置以及资源均显示在右边的“解决方案资源管理器”窗口中。



项目中的内容

属性

“属性”节点表示应用于整个项目的配置设置，这些设置存储在解决方案文件夹的 .csproj 文件中。这些设置包括编译选项、安全性和部署设置，以及其他更多设置。可以使用“项目设计器”对项目进行修改，该设计器是一组“属性页”，可以通过右击“属性”然后选择“打开”进行访问。有关更多信息，请参见[修改项目属性 \(Visual C#\)](#)。

引用

在项目的上下文中，引用仅标识应用程序运行所需的二进制文件。通常，引用标识 DLL 文件，如 .NET Framework 类库文件之一。它也可以引用 .NET 程序集(称为 shim)，使您的应用程序可以调用 COM 对象或本机 Win32 DLL 上的方法。如果您的程序创建了其他程序集中定义的类的实例，则必须在您的项目中添加对该文件的引用，才能编译项目。若要添加引用，请单击“项目”菜单上的“添加引用”。默认情况下，所有 C# 项目都包含对 mscorelib.dll 的引用，mscorelib.dll 包含核心 .NET Framework 类。可以通过单击“项目”菜单然后选择“添加引用”来添加对其他 .NET Framework DLL 和其他文件的引用。

注意

不要将项目引用的概念与 C# 或其他编程语言中的引用类型的概念相混淆。前者指文件及其在磁盘上的预期位置。后者指使用 **class** 关键字声明的 C# 类型。

资源

资源是应用程序中包含的数据，但是以可独立于其他源代码进行修改的方法存储。例如，您可以将所有字符串作为资源存储，而不是将它们硬编码到源代码中。您可以在日后将这些字符串翻译成不同语言，然后将它们添加到交付给客户的应用程序文件夹中，而不必重新编译程序集。Visual C# 定义五种类型的资源：字符串、图像、图标、音频和文件。可以使用“资源设计器”(可以在“项目设计器”中的“资源”选项卡上访问)添加、移除或编辑资源。

窗体

创建 Windows 窗体项目时，默认情况下，Visual C# 会将一个窗体添加到项目中，并为其命名为 Form1。表示该窗体的两个文件称为 Form1.cs 和 Form1.designer.cs。您可在 Form1.cs 中写入代码；designer.cs 文件是 Windows 窗体设计器写入代码的文件，这些代码用于实现所有通过从“工具箱”中拖放控件执行的操作。

可以通过单击“项目”菜单项然后选择“添加 Windows 窗体”来添加新窗体。每个窗体都有两个文件与其相关联。Form1.cs(您可以任意命名该文件)包含您写入的源代码，这些源代码用于配置窗体及其控件(如列表框和文本框)，并对事件(如单击按钮和按键)作出响应。在简单的 Windows 窗体项目中，需要在此文件中进行大部分或全部编写代码工作。

Designer.cs 文件包含将控件拖动到窗体、在“属性”窗口中设置属性等操作时“窗体设计器”写入的源代码。通常，根本不应该手动编辑此文件。

注意

显然,如果您创建控制台应用程序项目,它将不包含 Windows 窗体的源代码文件。

其他源代码文件

项目可以包含任意数量的其他 .cs 文件,这些文件可能与也可能不与特定的 Windows 窗体相关联。在上一个“解决方案资源管理器”图示中,program.cs 包含应用程序的入口点。单个 .cs 文件可以包含任意数量的类和结构定义。可以通过单击“项目”菜单上的“添加新项”或“添加现有项”将新的或现有的文件或类添加到项目中。

请参见

任务

[如何: 创建解决方案和项目生成配置](#)

[如何: 创建 Windows 应用程序项目](#)

概念

[介绍解决方案、项目和项](#)

[使用解决方案资源管理器](#)

[解决方案资源管理器中隐藏的项目文件](#)

[控制项目和解决方案](#)

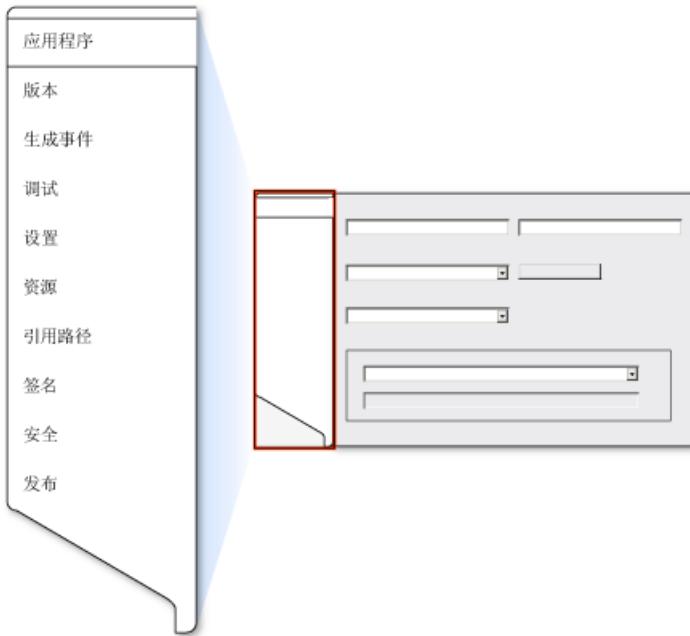
其他资源

[Visual C#](#)

[使用 Visual C# IDE](#)

修改项目属性 (Visual C#)

创建项目之后，可以使用“项目设计器”执行各种任务，例如更改可执行文件的名称、自定义生成过程、添加对 DLL 的引用或加强安全设置。单击“项目”菜单中的“属性”，或在“解决方案资源管理器”中右击“属性”项，可以访问“项目设计器”。“项目设计器”将显示在编辑器/设计器窗口，如下图所示：



在“项目设计器”中，对各种项目属性进行分组，包含在 10 个页面中。“项目设计器”属性页与“Windows 窗体设计器”和代码编辑器位于同一个中间窗格中。

注意

Visual Studio Team System 中包括用于代码分析的附加属性页。

在上面的图示中，显示了“应用程序”属性页。通过单击左侧选项卡上的各个标签（“生成”、“生成事件”、“调试”等），可以访问相应的属性页。此处输入的项目特定信息存储在 .csproj 文件中，该文件在“解决方案资源管理器”中不可见，但它位于驱动器上的项目文件夹中。使用 Visual C# 时，将鼠标光标定位在任意属性页上，然后按 F1，即可访问该属性页的帮助。

下表提供对“项目设计器”中各页的简短说明：

属性页	说明
应用程序	更改程序集的名称、项目类型及程序集信息（其中包括版本号和其他资源选项）。有关更多信息，请参见“ 项目设计器”->“应用程序”页 (C#) ”。
生成	更改已编译的程序集存储的位置、条件编译选项、错误和警告的处理方式以及其他设置。有关更多信息，请参见“ 项目设计器”->“生成”页 (C#) ”。
生成事件	创建和修改自定义生成步骤。有关更多信息，请参见“ 项目设计器”->“生成事件”页 (C# 和 J#) ”。
调试	指定在调试器下运行时的命令行参数，以及其他设置。有关更多信息，请参见“ 项目设计器”->“调试”页 ”。
资源	将字符串、图标、图像或其他类型的文件作为资源添加到项目中。有关更多信息，请参见“ 项目设计器”->“资源”页 ”。
设置	存储设置，如数据库的连接字符串，或特定用户要使用的配色方案。在运行时可以动态检索这些设置。有关更多信息，请参见“ 项目设计器”->“设置”页 ”。

引用路径	指定项目中引用的程序集所在位置的路径。有关更多信息,请参见 “项目设计器”->“引用路径”页(C#、J#) 。
签名	指定 ClickOnce 证书选项,并提供程序集的强名称。有关更多信息,请参见 “项目设计器”->“签名”页 和 ClickOnce 部署概述 。
安全性	指定运行应用程序所需的安全设置。有关更多信息,请参见 “项目设计器”->“安全”页 。
发布	指定将应用程序分发到网站、FTP 服务器、或文件位置所使用的选项。有关更多信息,请参见 “项目设计器”->“发布”页 。
代码分析(仅限 Visual Studio Team System)	用于分析源代码是否存在潜在安全问题、是否遵从 .NET Framework 设计指南,以及更多内容的工具选项。有关更多信息,请参见 “项目设计器”->“代码分析” 。

请参见

[概念](#)

[项目设计器介绍](#)

[其他资源](#)

[使用 Visual C# IDE](#)

[Visual Studio 集成开发环境](#)

设计用户界面 (Visual C#)

在 Visual C# 中, 创建用户界面 (UI) 最为快捷方便的方法就是使用“Windows 窗体设计器”和“工具箱”, 以直观方式创建。创建所有用户界面都有三个基本步骤:

- 将控件添加到设计图面。
- 设置控件的初始属性。
- 为指定事件编写处理程序。

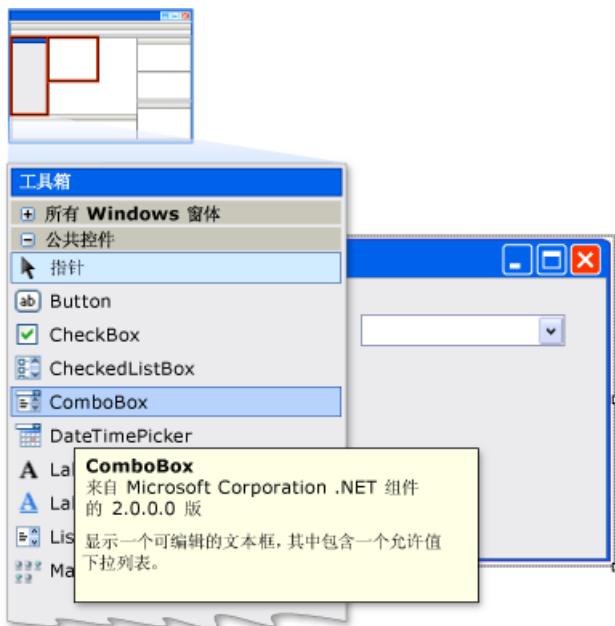
虽然也可以通过编写自己的代码创建 UI, 但使用各种设计器使您能够以较手动编码快得多的速度完成此项工作。

注意

也可以使用 Visual C# 创建控制台应用程序, 此类应用程序具有基于文本的简单 UI。有关更多信息, 请参见[创建控制台应用程序 \(Visual C#\)](#)。

添加控件

在设计器中, 可以使用鼠标将控件(如按钮和文本框)拖动到表示窗体的设计图面上。下图显示了一个已从“工具箱”窗口拖动到“Windows 窗体设计器”中某一窗体的组合框。



以直观方式进行工作时, 该设计器将您的操作转换成 C# 源代码, 并将这些代码写入称为 <name>.designer.cs 的项目文件中, 其中 <name> 是您为窗体提供的名称。应用程序运行时, 这些源代码将调整 UI 元素的位置和大小, 以使这些元素的外观与它们在设计图面上的外观一致。有关更多信息, 请参见[Windows 窗体设计器](#)。

设置属性

将控件添加到窗体后, 可以使用“属性”窗口设置其属性, 如背景色和默认文本。在“属性”窗口中指定的值仅为初始值, 在运行时创建控件时, 会将这些初始值赋给该属性。在很多情况下, 只需通过获取或设置应用程序中控件类的实例的属性, 即可在运行时以编程方式访问或更改这些值。“属性”窗口在设计时非常有用, 因为使用它可以浏览控件支持的所有属性、事件和方法。有关更多信息, 请参见[“属性”窗口](#)。

处理事件

具有图形用户界面的程序主要是由事件驱动的。在用户执行操作(如在文本框中输入文本、单击按钮或更改列表框中的选择)前, 这些程序会一直等待。当用户执行操作时, 控件(仅是 .NET Framework 类的一个实例)将向应用程序发送一个事件。您可以选择在应用程序中编写特定方法来处理事件, 在接收到事件时将调用此方法。

可以使用“属性”窗口指定在代码中要处理哪些事件;在设计器中选择一个控件, 然后单击“属性”窗口工具栏上带有闪电形图标“事件”按钮, 可查看该控件的事件。下图显示事件按钮。



通过“属性”窗口添加事件处理器时，设计器将自动编写空的方法体，需要您在其中编写相应的代码，使该方法执行有用的操作。大多数控件会生成大量事件，但在大多数情况下，应用程序只需处理其中某些或者仅仅一个事件。例如，您可能需要处理一个按钮的 **Click** 事件，但除非要以某种高级方式自定义其外观，否则无需处理该按钮的 **Paint** 事件。

后续步骤

有关 Windows 窗体用户界面的更多信息，请参见以下主题：

- [创建基于 Windows 的应用程序](#)
- [演练: 创建简单的 Windows 窗体](#)
- [Windows 窗体设计器用户界面元素](#)

在 .NET Framework 类库中，[System.Windows.Forms](#) 和相关的命名空间中包含进行 Windows 窗体开发所使用的类。

请参见

其他资源

[Visual C#](#)

[使用 Visual C# IDE](#)

编辑代码 (Visual C#)

Visual C# 代码编辑器是编写源代码的字处理器。就像 Microsoft Word 对句子、段落和语法提供广泛支持一样, C# 代码编辑器也为 C# 语法和 .NET Framework 提供广泛支持。这些支持可以分为五个主要的类别:

- IntelliSense: 将 .NET Framework 类和方法键入编辑器时, 不断对其基本文档进行更新, 同时还具有自动代码生成功能。
- 重构: 随着基本代码在开发项目过程中的演变, 智能重构基本代码。
- 代码段: 可以浏览的库, 其中包含了频繁重复的代码模式。
- 波浪下划线: 当您键入内容时, 对拼写错误的单词、错误的语法以及警告情况的可见通知。
- 可读性帮助: 大纲显示和着色。

IntelliSense

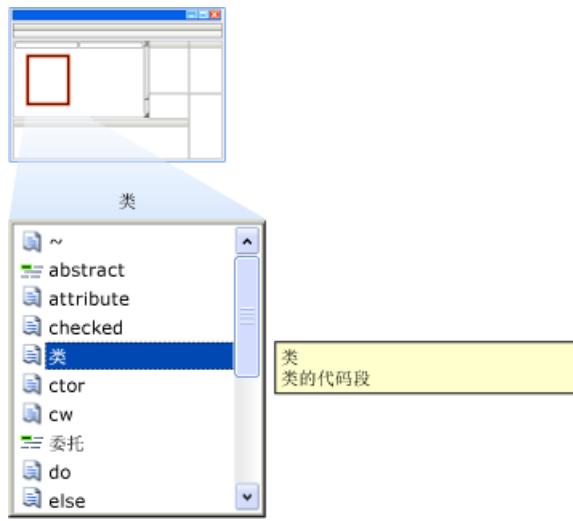
IntelliSense 是一组相关功能的名称, 旨在尽量减少查找帮助所需的时间, 有助于更加准确高效地输入代码。这些功能都提供了在编辑器中键入的语言关键字、.NET Framework 类型和方法签名的基本信息。这些信息会显示在工具提示、列表框和智能标记中。

注意

IntelliSense 中的很多功能都可以与其他 Visual Studio 语言共享, 并以举例说明的形式记录在 MSDN 库的[编码辅助工具](#)节点中。以下章节提供了对 IntelliSense 的简要概述, 还包括到更多完整文档的链接。

完成列表

在编辑器中输入源代码时, IntelliSense 将显示一个包含所有 C# 关键字和 .NET Framework 类的列表框。如果在列表框中找到了与正键入的名称相匹配的项, 将选择此项。如果选定的项就是您需要的项, 只需按 Tab, Intellisense 会完成名称或关键字的输入。有关更多信息, 请参见[C# 中的完成列表](#)。



快速信息

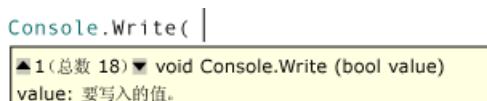
将光标悬停在一个 .NET Framework 类型上时, IntelliSense 将显示包含该类型基本文档的快速信息工具提示。有关更多信息, 请参见[快速信息](#)。

列出成员

将一个 .NET Framework 类型输入代码编辑器, 再键入点运算符 (.), IntelliSense 将显示包含该类型各成员的列表框。进行选择并按 Tab 后, IntelliSense 将输入该成员名称。有关更多信息, 请参见[列出成员](#)。

参数信息

在代码编辑器中输入方法名称, 再键入左括号后, IntelliSense 会显示参数信息提示工具, 其中显示了此方法的参数的顺序和类型。如果已重载此方法, 可以在所有已重载的签名中上下滚动进行查找。有关更多信息, 请参见[参数信息](#)。



添加 using

有时您可能会试图为名称并未充分限定的 .NET Framework 类创建实例。在此情况下，IntelliSense 会在无法解析的标识符后面显示智能标记。单击此智能标记时，IntelliSense 会显示 **using** 指令的列表，这些指令使该标识符可以被解析。IntelliSense 会将您从列表中选择的指令添加到源代码文件的顶部，您就可以在当前位置继续编码。有关更多信息，请参见[添加 using](#)。

重构

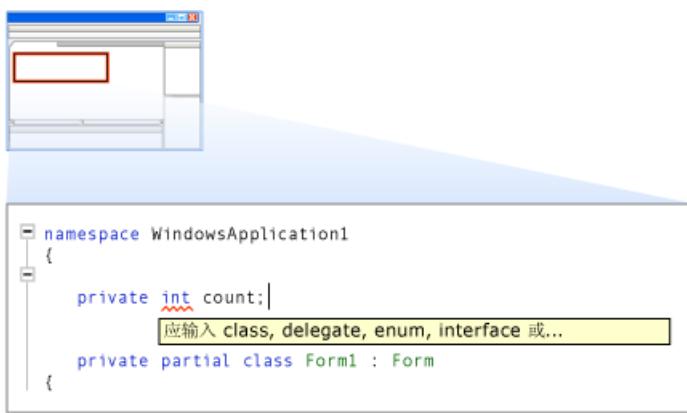
随着基本代码在开发项目过程中的不断增大和演变，有时需要进行更改，使其更具有可读性或可移植性。例如，您可能需要将一些方法拆分成更小的方法，或更改方法参数，或重命名标识符。在代码编辑器中右击就可访问重构功能，重构功能可以完成所有这些操作，比传统工具（如查找和替换）更便捷、更智能、更彻底。有关更多信息，请参见[重构](#)。

代码段

代码段是常用的 C# 源代码的小单元，只需几次击键便能准确快速地输入。在代码编辑器中右击，就能访问代码段菜单。可以在 Visual C# 提供的许多代码段中浏览，也可以创建自己的代码段。有关更多信息，请参见[代码段 \(C#\)](#)。

波浪下划线

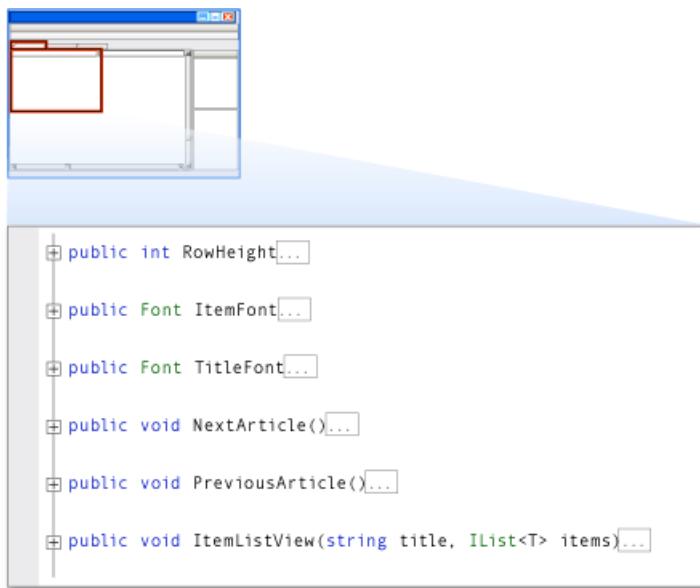
波浪下划线可以即时反馈键入代码时发生的错误。红色的波浪下划线标识语法错误，例如缺少分号或大括号不匹配。绿色的波浪下划线标识潜在的编译器警告，而蓝色的波浪下划线标识[编辑并继续](#)问题。下面的插图显示红色波浪下划线：



可读性帮助

大纲显示

代码编辑器会自动将命名空间、类和方法视为可折叠区域，以便于查找和读取源代码文件的其他部分。还可以在代码周围添加 **#region** 和 **#endregion** 指令，创建自己的可折叠区域。



着色

在 C# 源代码文件中，编辑器对不同类别的标识符使用不同的颜色。有关更多信息，请参见[代码着色](#)。

请参见

其他资源

[使用 Visual C# IDE](#)

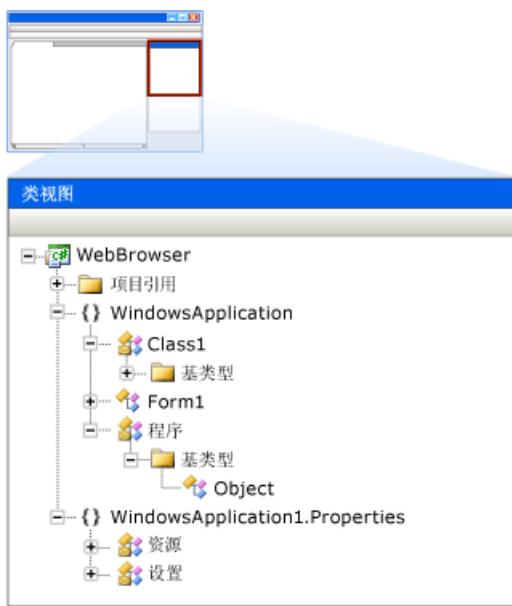
导航和搜索 (Visual C#)

Visual C# 提供了以下工具，帮助您在源代码、项目文件和打开窗口中进行导航和搜索。

- 类视图
- 导航栏
- Ctrl-Tab 导航
- 在文件中查找

类视图

“类视图”窗口提供项目视图(基于类而不是文件)，如同“解决方案资源管理器”中一样。可以使用“类视图”快速导航到项目中的任何类或类成员。若要访问“类视图”，请在“视图”菜单中单击“类视图”。



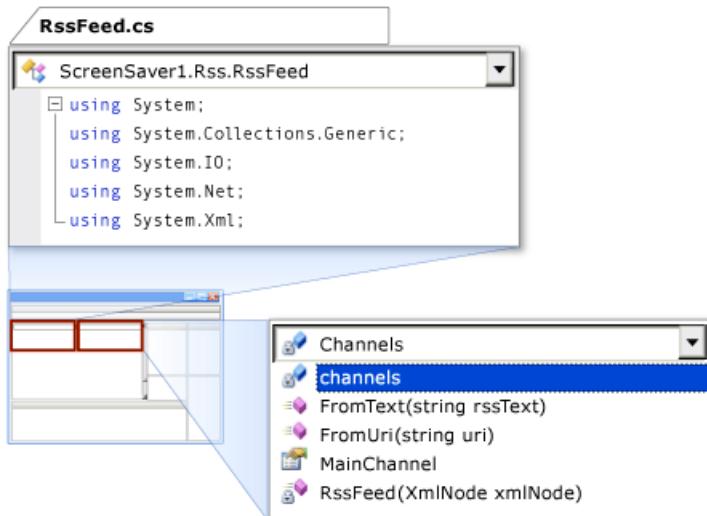
Ctrl-Tab 导航

在任何给定时间，在 Visual C# 项目中都可以拥有多个活动窗口。若要快速导航到某个窗口，请按 Ctrl+Tab，显示一个列出所有活动工具和源代码窗口的窗口。按住 Ctrl，同时移动箭头键，选择要显示的窗口。



导航栏

在每个代码编辑器窗口的顶部是导航栏，它由两个列表框组成。左侧的列表框列出了当前文件中定义的全部类，右侧则列出了左侧列表框中选定类的全部成员。在右侧列表框中选择某个方法，就可以直接转到该方法。



在文件中查找

按 Ctrl+Shift+F 可以打开“在文件中查找”对话框，从而在整个项目中执行查找和替换操作。

注意

若要重命名方法或类型，或更改方法参数，请使用重构功能，它比查找和替换更彻底、更智能化。有关更多信息，请参见[重构](#)。

有关更多信息

- [如何：定位代码和文本](#)
- [如何：显示代码大纲和隐藏代码](#)
- [如何：对文档进行渐进式搜索](#)

请参见

其他资源

[Visual C#](#)

[使用 Visual C# IDE](#)

生成和调试 (Visual C#)

在 Visual C# 中，可以在“生成”菜单中单击“生成”（或按 Ctrl+Shift+B），生成可执行应用程序。按 F5 或在“调试”菜单中单击“运行”，可以通过一步操作生成并启动应用程序。

生成过程包括将项目文件输入到 C# 编译器，将源代码转化为 Microsoft 中间语言 (MSIL)，然后将 MSIL 和元数据、资源、清单以及其他模块（如果有的话）进行联接，从而创建一个程序集。程序集是一个可执行文件，扩展名通常为 .exe 或 .dll。开发应用程序时，有时会希望生成调试版本，以进行测试并查看程序的运行状况。最后，在一切无误的情况下，将创建一个发布版本部署给客户。

有关程序集的更多信息，请参见[程序集概述](#)。

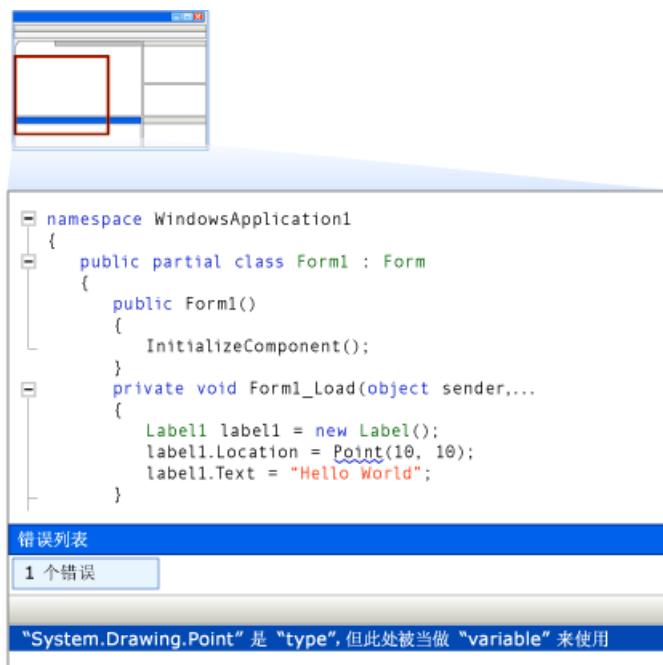
生成设置

若要指定不同的生成设置，请在“解决方案资源管理器”中右击项目项，然后在“项目设计器”中选择“生成”窗格。有关更多信息，请参见[项目设计器介绍](#)和[C# 编译器选项](#)。

Visual Studio 使用 MSBuild 工具创建程序集。也可以从命令行运行 MSBuild，并可通过多种方式进行自定义。有关更多信息，请参见[MSBuild](#)。

生成错误

如果 C# 语法中存在错误，或存在无法解析为已知类型或成员的标识符，生成将失败，并在“[错误列表](#)”窗口中出现一个错误列表，默认情况下该窗口直接出现在代码编辑器下方。可以双击错误信息，转到发生错误的代码行。



通常 C# 编译器的错误信息会非常明确，并且具有较强的描述性，但如果无法断定问题所在，可以在错误列表中选中该错误信息，并按 F1，转到该信息对应的“帮助”页面。“帮助”页面中还包含其他的有用信息。如果问题仍然无法解决，则下一步应当在 C# 论坛或新闻组中提出该问题。若要访问论坛，请在“社区”菜单上单击“提出问题”。

注意

如果您遇到了一个编译器错误，但此错误的“帮助”页面未能提供有用的帮助，您可以发送该问题的说明，帮助 Microsoft 改进文档。若要发送电子邮件，请单击包含此错误的“帮助”页面底部的链接。

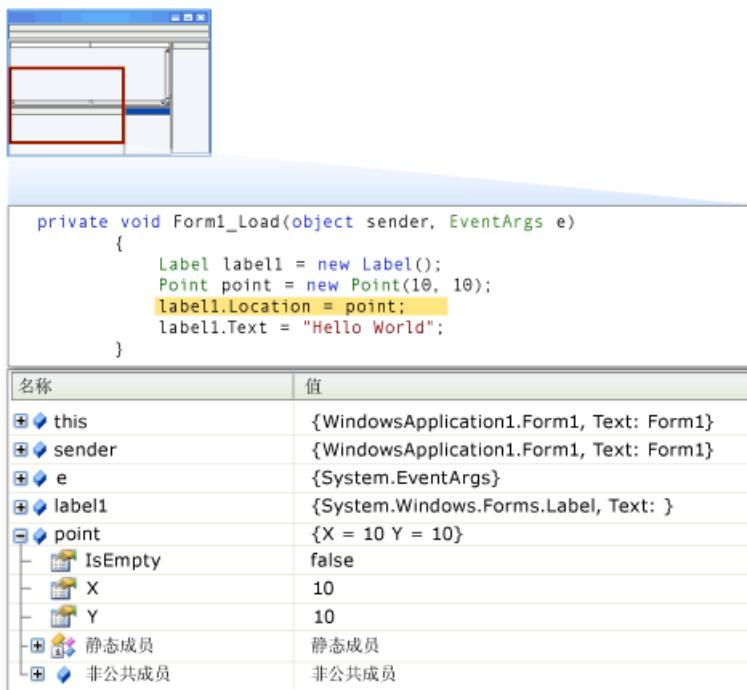
发布与调试配置

在您仍然忙于项目时，通常会使用调试配置生成应用程序，因为这一配置可用于在调试器中查看变量的值和控制执行情况。还可以用发布配置创建并测试各版本，以确保没有引入任何仅在某一种类型的版本中出现的 bug。在 .NET Framework 编程中，此类 bug 非常少见，但它们还是有可能发生的。

当您准备将应用程序分发至最终用户时，可以创建一个发布版本。与相应的调试配置相比，它的大小很小，且通常性能更好。可以在“项目设计器”的“生成”窗格或“生成”工具栏中设置生成配置。有关更多信息，请参见[生成配置](#)。

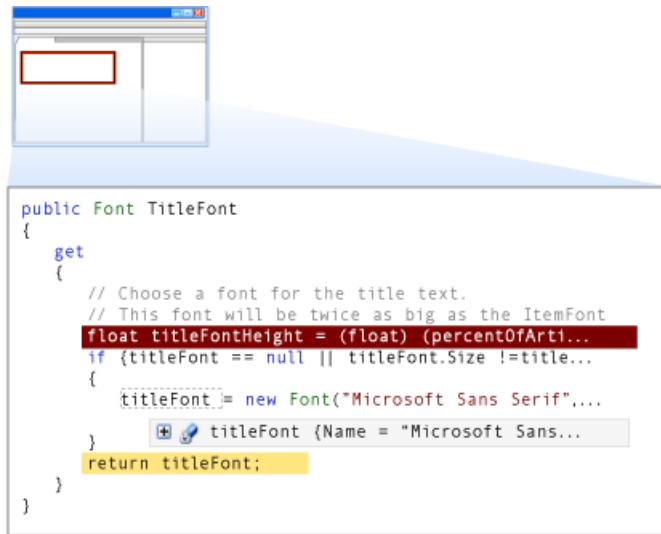
调试

在任何时候使用代码编辑器时，都可以按 F9 在代码的某一行设置断点。在 Visual Studio 调试器中按 F5 运行应用程序时，应用程序会在该行停止，此时可以检查任何给定变量的值，或观察执行跳出循环的时间和方式，按 F10 逐行单步执行代码，或设置其他断点。



还可以设置条件断点，仅在满足指定条件的情况下才会停止执行。跟踪点类似于断点，区别在于它并不停止执行，只是将指定变量的值写入输出窗口。有关更多信息，请参见[断点与跟踪点](#)。

当执行在某个断点处停止时，可以将鼠标悬停于该范围内的任何变量上，以查看有关该变量的信息。下图显示了调试器中的一个数据提示：



调试器在断点处停止后，可以按 F10 逐行单步执行代码。甚至可以修复代码中某些类型的错误并继续调试，无需停下来重新编译应用程序。

Visual Studio 调试器是一个功能强大的工具，花时间阅读该文档以理解不同的概念，如[编辑并继续](#)、[查看调试器中的数据](#)、[可视化工具](#)和[实时调试](#)是值得的。

请参见

任务

[如何：设置调试和发布配置](#)

[如何：在编辑器中调试代码](#)

参考

[System.Diagnostics](#)

[其他资源](#)

[Visual C#](#)

[使用 Visual C# IDE](#)

[调试准备:C#、J# 和 Visual Basic 项目类型](#)

[调试设置和准备](#)

建模和分析代码 (Visual C#)

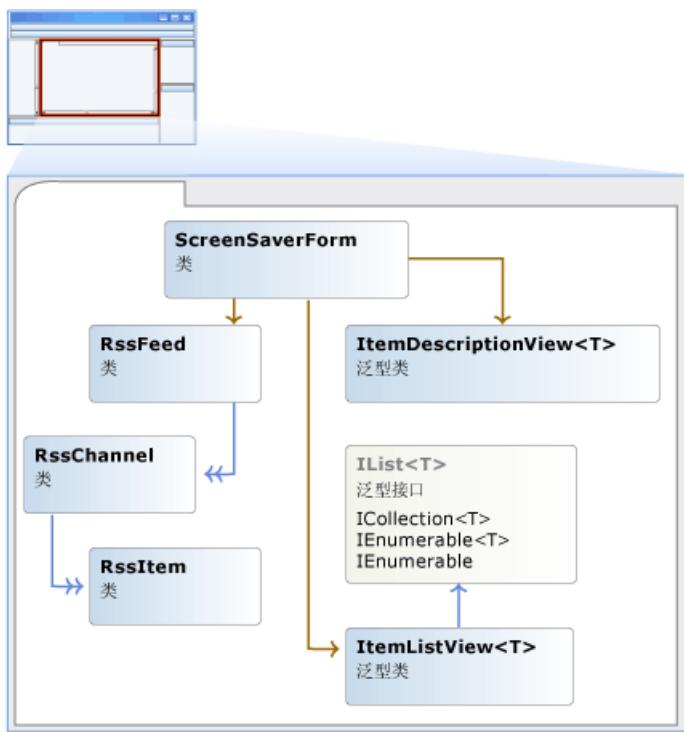
对于软件开发人员来说，经常对所处理的源代码基本结构不熟悉，这是因为此代码是由其他人编写的，或者此代码是很久以前编写的，而最初的编写者已记不起此代码的工作原理。另一种常见的情况是需要理解只能以二进制格式提供的库的内容。Visual C# 提供了以下工具，可以帮助您建模、分析和理解源代码和二进制程序集中的类型以及类型关系：

- 类设计器，用于以直观方式表示类型之间的继承和关联关系。
- 对象浏览器，用于检查由 .NET Framework 程序集以及包括 COM 对象的本机 DLL 导出的类型、方法和事件。
- 用作源代码的元数据，用于查看托管程序集中的类型信息，查看方式与查看项目中的源代码的方式相同。

除上面列出的工具外，Visual Studio Team System 还包括托管代码代码分析工具，用于检查您的代码是否存在各种类型的潜在问题。

类设计器

类设计器是一种图形工具，用于以直观方式对软件应用程序或组件中各类型之间的关系进行建模；也可以使用此工具设计新类型和重构或删除现有类型。下图显示了一个简单的类设计：

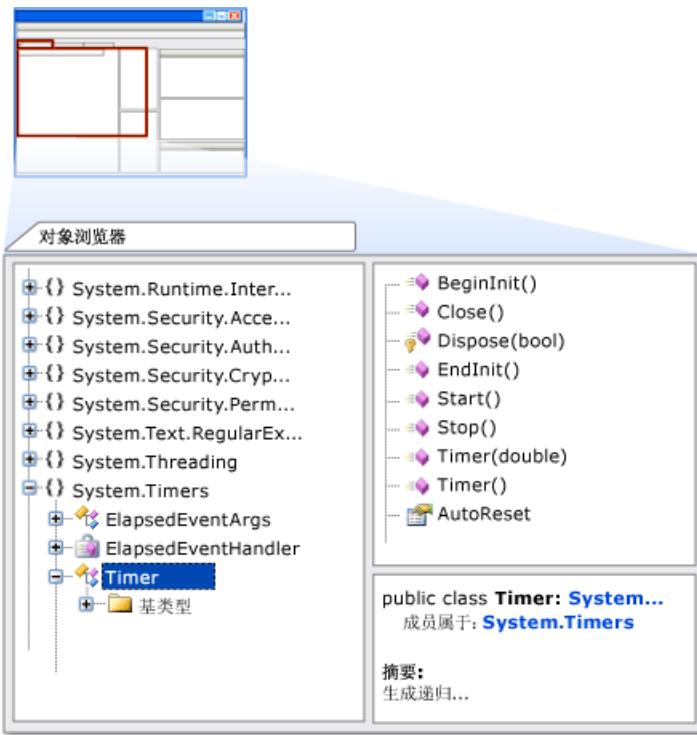


若要将类关系图添加到项目中，请单击“项目”菜单上的“添加新项”，然后单击“添加类关系图”。

有关更多信息，请参见[设计和查看类与类型](#)。

对象浏览器

对象浏览器使您可以查看本机和托管 DLL（包括 COM 对象）中的类型信息。虽然对象浏览器中显示的信息与类视图中显示的信息相似，但您可以使用对象浏览器检查系统中的所有 DLL，而不仅仅是自己的项目中引用的 DLL。此外，对象浏览器还显示选定类型的 XML 文档注释。下图显示了对象浏览器如何显示二进制文件中的类型信息。



有关更多信息，请参见[对象浏览器](#)

用作源代码的元数据：

用作源代码的元数据功能使您可以查看托管程序集中类的类型信息，查看方式与查看项目中源代码的方式相同。这种方法非常方便，当您无权访问实际源代码时，也可以一目了然地查看类中所有公共方法的签名。

例如，如果您在代码编辑器中输入语句 `System.Console.WriteLine()`，将插入点置于 `Console` 中，然后右击并选择“转到定义”，将会看到类似源代码文件的内容，其中包含 `Console` 类的声明。此声明是使用[反射](#)从程序集中的元数据构造的，虽然此声明不公开任何方法的实现，但它确实能显示存在的所有 XML 文档注释。

也可以在对象浏览器中选择一个托管类型，然后单击“视图”菜单上的“代码定义窗口”，使用用作源代码的元数据功能。

有关更多信息及图示，请参见[作为源代码的元数据](#)。

托管代码代码分析

托管代码代码分析工具可以分析托管程序集并报告信息（如潜在的安全问题，以及 Microsoft .NET Framework 设计指导原则中阐明的编程与设计规则的冲突）。此信息以警告的形式出现。右击“解决方案资源管理器”中的“属性”，然后选择“打开”，可以访问“项目设计器”中的该工具。

有关更多信息，请参见[“项目设计器”->“代码分析”和托管代码的代码分析概述](#)。

请参见

[概念](#)

[编辑代码 \(Visual C#\)](#)

[反射 \(C# 编程指南\)](#)

[其他资源](#)

[使用 Visual C# IDE](#)

[类库开发的设计准则](#)

[异常设计准则](#)

[成员设计准则](#)

[类型设计准则](#)

添加和编辑资源 (Visual C#)

Visual C# 应用程序中经常会包含非源代码的数据。此类数据称为“项目资源”，它可以包含应用程序所需的二进制数据、文本文件、音频或视频文件、字符串表、图标、图像、XML 文件或任何其他类型的数据。项目资源数据以 XML 格式存储在 .resx 文件中（默认文件名为 Resources.resx），可在“解决方案资源管理器”中打开此文件。有关项目资源的更多信息，请参见[使用资源文件](#)。

向项目中添加资源

可以将资源添加到项目中，方法是：在“解决方案资源管理器”中，在该项目下右击“属性”节点，单击“打开”，再单击“项目设计器”中“资源”页上的“添加资源”按钮。

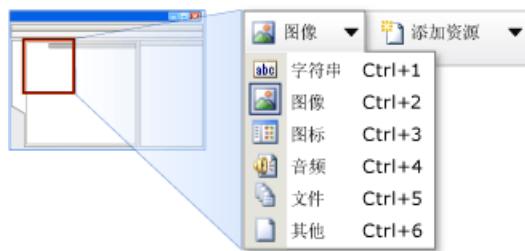
可以将资源作为链接的资源（外部文件）或嵌入的资源（直接嵌入到 .resx 文件中）添加到项目中。

- 在添加链接的资源时，存储项目资源信息的 .resx 文件仅包含指向磁盘上资源文件的相对路径。如果将图像、视频或其他复杂文件作为链接的资源进行添加，则可以使用默认编辑器编辑它们，可在资源设计器中为文件类型设置关联的默认编辑器。
- 当添加嵌入的资源时，数据直接存储到项目的资源 (.resx) 文件中。字符串只能作为嵌入的资源进行存储。

有关更多信息，请参见[链接的资源与嵌入的资源](#)和[.Resx 文件格式中的资源](#)。

编辑资源

关联用于编辑各个资源的默认应用程序之后，资源设计器就允许您在开发过程中添加和修改项目资源。右击“解决方案资源管理器”中的“属性”，然后单击“打开”，再单击项目设计器中的“资源”选项卡，可以访问资源设计器。有关更多信息，请参见[“项目设计器” -> “资源”页](#)。下面的图示显示了资源设计器菜单选项：



若要编辑嵌入的资源，必须直接处理 .resx 文件来操作每个字符或字节。这就是为什么在开发过程中将复杂文件类型存储为链接的资源更为方便的原因。可以使用[二进制编辑器](#)以十六进制或 ASCII 格式在二进制级别上编辑资源文件（包括 .resx 文件）。可以使用[图像编辑器](#)编辑作为链接的资源存储的图标、光标以及 jpeg 和 GIF 文件。您还可以选择使用其他应用程序作为这些文件类型的编辑器。有关更多信息，请参见[在资源编辑器中查看和编辑资源](#)。

将资源编译为程序集

当您生成应用程序时，Visual Studio 会调用 resgen.exe 工具，将应用程序资源转换为称为 Resources 的内部类。此类包含在 Resources.Designer.cs 文件中，而该文件嵌套在“解决方案资源管理器”中的 Resources.resx 文件下。Resources 类将所有项目资源封装到静态只读的 **get** 属性中，作为在运行时提供强类型资源的方式。当通过 Visual C# IDE 进行生成时，所有封装的资源数据（包括嵌入到 .resx 文件中的资源和链接的文件）都被直接编译到应用程序程序集 (.exe 或 .dll 文件) 中。换句话说，Visual C# IDE 总是使用 **/resource** 编译器选项。如果从命令行进行生成，则可以指定 **/linkresource** 编译器选项，以便将资源部署为主应用程序程序集之外的单独文件。这是一种高级应用方案，仅在某些很少见的情况下才需要使用。将资源与主应用程序程序集分开部署的一种更为常见的方案是使用附属程序集，如下所述。

在运行时访问资源

若要在运行时访问资源，只需像引用任何其他类成员一样引用该资源即可。下面的示例显示如何检索名为 Image01 的位图资源。请注意，资源类位于名为 <项目名称>.Properties 的命名空间中，因此，要么每个资源都使用完全限定名，要么在作为资源类访问来源的源文件中添加适当的 **using** 指令。

```
System.Drawing.Bitmap bitmap1 = myProject.Properties.Resources.Image01;
```

在内部，**get** 属性使用 **ResourceManager** 类创建对象的新实例。

有关更多信息，请参见[应用程序中的资源](#)和[资源文件生成器 \(Resgen.exe\)](#)。

附属程序集中的资源

如果您正在创建需要本地化（翻译）为多种语言的应用程序，可以将每组特定于区域性的字符串作为资源存储在它本身的附属程

序集中。当您分发应用程序时，其中应包含主应用程序程序集和任何相应的附属程序集。然后您可以在不重新编译主应用程序程序集的情况下，添加其他附属程序集或修改现有的附属程序集。有关更多信息，请参见[创建附属程序集](#)和[定位和使用特定区域性的资源](#)。

请参见

概念

[项目设计器介绍](#)

其他资源

[Visual C#](#)

[Visual C# 入门](#)

[公共语言运行库中的程序集](#)

[对应用程序进行全球化和本地化](#)

获得帮助 (Visual C#)

Visual Studio 的“帮助”文档包含在 MSDN Library 中，您可以将 MSDN Library 安装在本地计算机或网络上，也可以在 Internet 上的 <http://www.microsoft.com/china/msdn/library> 位置获得。该库的本地版本是格式为 .hxs 的压缩 HTML 文件的集合。您可以选择在计算机上安装该库的全部或部分内容；MSDN 完全安装的大小接近 2GB，并且其中包括很多 Microsoft 技术文档。使用称为 Microsoft 文档资源管理器的 Visual Studio 帮助浏览器可以查看本地和联机 MSDN 文档。

在使用 Visual C# 的过程中，有六种方法可以访问“帮助”：

- F1 搜索
- 搜索
- 索引
- 目录
- 如何实现
- 动态帮助

联机与本地帮助

在“选项”菜单下的“帮助选项”属性页上，可以指定搜索行为（包括 F1 搜索）的下列选项：

- 首先尝试搜索联机 MSDN library，如果未找到匹配项，再尝试搜索本地文档。
- 首先尝试搜索本地 MSDN library，如果未找到匹配项，再尝试搜索联机文档。
- 只尝试搜索本地 MSDN library。

第一次调用任何搜索时，都会出现这些选项。联机 MSDN 文档可能比本地文档包含更多更新内容。因此，如果在使用 Visual C# 的过程中可以连接 Internet，建议您将搜索选项设置为首先尝试搜索联机 MSDN library。本地文档一旦更新即可下载。有关文档更新的更多信息，请查看 [Visual Studio Developer Center \(Visual Studio 开发人员中心\)](#)。

F1 搜索

F1 提供区分上下文的搜索功能。在代码编辑器中，将插入点光标定位在关键字或类成员上或紧随其后，并按 F1，可以访问 C# 关键字和 .NET Framework 类的“帮助”文档。当对话框或任何其他窗口具有焦点时，可以按 F1 获取该窗口的“帮助”。

F1 搜索仅返回一页。如果没有找到匹配项，将显示信息性页面，提供一些故障排除提示。

搜索

使用搜索界面，返回与任何指定的术语或术语集相匹配的所有文档。

搜索界面的外观如下所示：



也可以使用“选项”菜单上的“帮助选项”页面，指定除了搜索 MSDN library 以外，是否还要搜索 Codezone 网站。Codezone 站点由 Microsoft 合作伙伴运作，这些站点提供有关 C# 和 .NET Framework 的有用信息。Codezone 内容只可以联机使用。

相同的联机与本地搜索选项可应用于搜索和 F1 搜索。

在“搜索”界面中，指定要包括的文档类型，可以缩小或扩大搜索范围。此处有三个选项：语言、技术和主题类型。通常情况下，只选中适用于当前开发方案的那些选项，将获得最佳效果。

索引

索引可以快速找到本地 MSDN library 中的文档。它不是全文搜索；而是只搜索分配给每个文档的索引关键字。索引查找通常比全文搜索更快而且相关性更强。如果不止一个文档中包含了索引搜索框中指定的索引关键字，将打开歧义消除窗口，您可以从可能的选项中进行选择。

默认情况下，“索引”窗口位于文档资源管理器的左侧。可以从 Visual C# 中的“帮助”菜单访问该窗口。

目录

MSDN library 目录以分层树视图结构显示库中的所有主题。它非常有用，可以浏览文档、了解库中所包含的内容，还可以浏览无法通过索引或搜索找到的文档。通常情况下，通过 F1、索引或搜索查找文档时，知道文档在目录中的位置十分有用，这样就可以查看存在哪些与给定主题相关联的其他文档。单击文档资源管理器工具栏中的“与目录同步”按钮，可查看当前显示的页面在 MSDN library 中的位置。

如何实现

“如何实现”是 MSDN library 的筛选视图，其中主要包括称为“如何”或“演练”的文档，这些文档说明如何执行特定任务。可以从“文档资源管理器”工具栏、“帮助”菜单或“开始”页访问“如何实现帮助”。Visual Studio 的每种语言都有其各自的“如何实现”页面，所显示的页面取决于当前的活动项目类型。

动态帮助

“动态帮助”窗口根据代码编辑器中插入点的当前位置，显示到 .NET Framework 和 C# 语言的参考文档的链接。有关更多信息，请参见 [如何：自定义动态帮助](#)。

请参见

其他资源

[使用 Visual C# IDE](#)

部署 C# 应用程序

部署是分发要安装到其他计算机上的已完成应用程序或组件的过程。对于控制台应用程序或基于 Windows 窗体的智能客户端应用程序，有两个部署选项可供选择：ClickOnce 和 Windows Installer。

ClickOnce 部署

ClickOnce 部署允许您将 Windows 应用程序发布到 Web 服务器或网络文件共享，以简化安装。在大多数情况下，建议使用 ClickOnce 选项进行部署，因为该选项可使基于 Windows 的应用程序进行自更新，尽可能减少安装和运行时所需的用户交互。

若要配置 ClickOnce 部署的属性，可以使用[发布向导](#)（可从“生成”菜单访问）或“项目设计器”中的“发布”页。有关更多信息，请参见[“项目设计器”->“发布”页](#)。有关 ClickOnce 的更多信息，请参见[ClickOnce 部署](#)。

Windows Installer

Windows Installer 部署允许您创建安装程序包以分发给用户；用户运行安装文件并按照向导逐步操作即可安装应用程序。将安装项目添加到解决方案中即可完成此操作；在生成后，它将创建一个分发给用户的安装文件；用户运行此安装文件并按照向导逐步操作即可安装应用程序。

有关 Windows Installer 的更多信息，请参见[Windows Installer 部署](#)。

请参见

任务

[如何：发布 ClickOnce 应用程序](#)

概念

[其他部署方法](#)

[ClickOnce 部署概述](#)

[使用 Windows Installer 部署运行库应用程序](#)

安装项目

其他资源

[Visual C#](#)

如何 : 向 C# 项目添加应用程序配置文件

通过向 C# 项目添加应用程序配置文件 (app.config 文件), 可以自定义公共语言运行库定位和加载程序集文件的方式。有关应用程序配置文件的更多信息, 请参见[运行库如何定位程序集](#)。

当生成项目时, 开发环境会自动创建 app.config 文件的副本并更改其文件名, 使其与可执行文件同名, 然后将新的 .config 文件移动到 bin 目录下。

向 C# 项目添加应用程序配置文件

1. 在“项目”菜单上单击“添加新项”。

随即显示“添加新项”对话框。

2. 选择“应用程序配置文件”模板, 然后单击“添加”。

名为 app.config 的文件被添加到您的项目中。

请参见

任务

[如何 : 使用应用程序配置文件指定 .NET Framework 的版本](#)

Visual C# 代码编辑器功能

Visual C# 提供了可以帮助您编辑和导航代码的工具。

本节内容

[重构](#)

列出可帮助您在不更改应用程序行为的前提下修改代码的重构操作。

[代码段 \(C#\)](#)

提供在 Visual C# 中使用代码段自动向应用程序中添加通用代码构造的概述。

[代码着色](#)

描述“代码编辑器”如何为代码构造着色。

[作为源代码的元数据](#)

描述在 IDE 中如何以源代码的形式查看元数据。

相关章节

[编辑文本、代码和标记](#)

描述如何使用“代码编辑器”编写代码并设置代码格式。

[浏览代码](#)

提供相关链接，通过这些链接可以找到使用“查找和替换”窗口、“书签”、“任务列表”和“错误列表”查找代码行的过程。

[Visual C# IDE 设置](#)

为 C# 开发人员提供 Visual Studio 设置的概述。

重构

重构是在编写代码后在不更改代码的外部行为的前提下通过更改代码的内部结构来改进代码的过程。

Visual C# 在“重构”菜单上提供了以下重构命令：

- [提取方法](#)
- [重命名](#)
- [封装字段](#)
- [提取接口](#)
- [将局部变量提升为参数](#)
- [移除参数](#)
- [重新排列参数](#)

多项目重构

Visual Studio 支持多项目重构。更正文件间的引用的所有重构操作也会更正同一语言的所有项目间的引用。这适用于所有项目间的引用。例如，如果具有一个引用类库的控制台应用程序，则当您重命名类库类型（使用 **Rename** 重构操作）时，也将更新该控制台应用程序中对类库类型的引用。

“预览更改”对话框

许多重构操作都可提供这样的机会：可以在提交引用更改之前检查重构操作将对代码执行的所有引用更改。对于这些重构操作，将在重构对话框中显示“预览引用更改”选项。选择该选项并接受重构操作后，将显示“[预览更改”对话框](#)。请注意，“预览更改”对话框具有两个视图。底部视图将显示代码，其中包含了因重构操作而引起的所有引用更新。按下“预览更改”对话框中的“取消”将停止重构操作，并且代码不会进行任何更改。

容错重构

重构可以容错。换句话说，可以在无法生成的项目中执行重构。但是，在这些情况下，重构过程可能不会正确更新不明确的引用。

请参见

任务

[如何：还原 C# 重构代码段](#)

其他资源

[Visual C# 代码编辑器功能](#)

提取方法

Extract Method 是一项重构操作，提供了一种从现有成员中的代码段创建新方法的便捷方式。

使用 **Extract Method**，可以通过从现有成员的代码块中提取选定的代码来创建新方法。创建的新方法中包含选定的代码，而现有成员中的选定代码被替换为对新方法的调用。将代码段转换为其自己的方法，使您可以快速而准确地重新组织代码，以获得更好的重用和可靠性。

Extract Method 具有以下优点：

- 通过强调离散的可重用方法鼓励最佳的编码做法。
- 鼓励通过较好的组织获得自记录代码。当使用描述性名称时，高级别方法可以像读取一系列注释一样进行读取。
- 鼓励创建细化方法，以简化重载。
- 减少代码重复。

备注

使用“提取方法”命令时，将在同一个类中的源成员之后插入新方法。

分部类型

如果类是分部类型，则 **Extract Method** 将紧跟源成员之后生成新方法。**Extract Method** 确定新方法的签名，并在新方法中的代码不引用实例数据时创建静态方法。

泛型类型参数

当您提取的方法具有不受约束的泛型类型参数时，除非已为该参数赋值，否则生成的代码将不向该参数添加“ref”修饰符。如果提取的方法将支持引用类型作为泛型类型参数，则应该为方法签名中的参数手动添加“ref”修饰符。

匿名方法

如果您尝试提取匿名方法的一部分（包含对在匿名方法之外声明或引用的局部变量的引用），则 Visual Studio 将警告您可能有语义更改。特别需要指出的是，将局部变量的值传递给匿名方法的时间将有所不同。

当匿名方法使用局部变量的值时，将在执行匿名方法时获取该值。将匿名方法提取到其他方法中时，将在调用提取方法时获取局部变量的值。

下面的示例阐释这一语义更改。如果执行此代码，则将向控制台输出 **11**。如果使用“提取方法”来提取代码注释在其自己的方法中标记的代码区域，然后执行重构后的代码，则将向控制台输出 **10**。

```
class Program
{
    delegate void D();
    D d;
    static void Main(string[] args)
    {
        Program p = new Program();
        int i = 10;
        /*begin extraction*/
        p.d = delegate { Console.WriteLine(i++); };
        /*end extraction*/
        i++;
        p.d();
    }
}
```

若要解决此问题，请使匿名方法中使用的局部变量成为类的字段。

请参见

任务

[如何：使用“提取方法”重构代码](#)

“提取方法”对话框

使用此对话框可以指定“提取方法”将生成的新方法的名称，并验证“提取方法”是否将按照您的意愿构造新方法签名。

若要访问此对话框，请从“重构”菜单中选择“提取方法”命令。仅当代码选择对于提取方法操作有效时，此对话框才可用。

新方法名称

键入“提取方法”将生成的新方法的名称。

预览方法签名

显示新方法签名的预览。该签名由重构引擎基于选定代码自动确定，并且不能在此文本框中进行修改。

请参见

任务

[如何：使用“提取方法”重构代码](#)

参考

[提取方法](#)

如何：使用“提取方法”重构代码

下面的过程描述如何从现有成员的代码段来创建新的方法。使用此过程可以执行[提取方法](#)重构操作。

使用“提取方法”

- 按下面示例中所述创建控制台应用程序。

有关更多信息，请参见[控制台应用程序](#)。

- 在[代码编辑器](#)中选择要提取的代码段：

```
double area = PI * radius * radius.
```

- 在“重构”菜单中选择“提取方法”。随即显示“[提取方法](#)”对话框。

您还可以键入键盘快捷键 Ctrl+R、Ctrl+M 来显示“提取方法”对话框。

还可以右击选定的代码，指向上下文菜单中的“重构”，然后单击“提取方法”以显示“提取方法”对话框。

- 在“新方法名称”文本框中指定新方法的名称，如 **CircleArea**。新方法签名的预览显示在“预览方法签名”下。
- 单击“确定”按钮。

示例

若要设置此示例，请创建一个名为 ExtractMethod 的控制台应用程序，然后使用以下代码替换 Class1。有关更多信息，请参见[控制台应用程序](#)。

```
class A
{
    const double PI = 3.141592;

    double CalculatePaintNeeded(double paintPerUnit, double radius)
    {
        // Select any of the following:
        // 1. The entire next line of code.
        // 2. The right-hand side of the next line of code.
        // 3. Just "PI *" of the right-hand side of the next line
        //     of code (to see the prompt for selection expansion).
        // 4. All code within the method body.
        // ...Then invoke Extract Method.

        double area = PI * radius * radius;

        return area / paintPerUnit;
    }
}
```

请参见

[参考](#)

[提取方法](#)

[概念](#)

[重构](#)

重命名

Rename 是一项重构操作，提供了一种重命名代码符号（如字段、局部变量、方法、命名空间、属性和类型）标识符的简单方法。“重命名”功能除了可用来更改标识符的声明和调用以外，还可用来更改注释中和字符串中的名称。

注意

在使用 [用于 Visual Studio 的源代码管理](#)时，请在尝试执行重命名重构功能前，获取最新版本的源文件。

通过以下 Visual Studio 功能可以使用重命名重构功能：

功能	开发环境中的重构行为
代码编辑器	在“代码编辑器”中，将光标置于代码符号声明中时，可以使用重命名重构功能。当光标位于此位置时，可以采用以下两种方式调用“重命名”命令：键入键盘快捷键，或者从智能标记、上下文菜单或“重构”菜单中选择“重命名”菜单项。选择“重命名”菜单项时，将显示“重命名”对话框。有关更多信息，请参见 “重命名”对话框 和 如何：重命名标识符 。
类视图	在“类视图”中选择标识符时，可以从上下文菜单和“重构”菜单中使用重命名重构功能。
对象浏览器	在“对象浏览器”中选择标识符时，只能在“重构”菜单中使用重命名重构功能。
Windows 窗体设计器的“属性网格”	在 Windows 窗体设计器的“属性网格”中，更改控件名称将启动该控件的重命名操作。不会显示“重命名”对话框。
解决方案资源管理器	<p>在“解决方案资源管理器”中，可以在上下文菜单中使用“重命名”命令。如果选定的源文件包含类名与文件名相同的类，则可以使用此命令同时重命名源文件并执行重命名重构。</p> <p>例如，如果创建默认的 Windows 应用程序，然后将 Form1.cs 重命名为 TestForm.cs，则源文件名 Form1.cs 将更改为 TestForm.cs，并且 Form1 类以及对该类的所有引用都将重命名为 TestForm。</p> <p>注意</p> <p>“撤消”命令 (Ctrl+Z) 将仅撤消代码中的重命名重构操作，而不会将文件名改回原始名称。</p> <p>如果选定的源文件不包含类名与文件名相同的类，则“解决方案资源管理器”中的“重命名”命令将仅重命名源文件，而不会执行重命名重构。</p>

重命名操作

执行 **Rename** 时，重构引擎将执行特定于下表中描述的每种代码符号的重命名操作。

代码符号	重命名操作
字段	将字段的声明和用法更改为新名称。
局部变量	将变量的声明和用法更改为新名称。
方法	将方法的名称以及对该方法的所有引用更改为新名称。

命名空间	将声明、所有正在使用的语句及完全限定名称中的命名空间名称更改为新名称。
	<p> 注意</p> <p>重命名命名空间时, Visual Studio 还更新“项目设计器”的“应用程序页”上的“默认命名空间”属性。此属性不能通过选择“编辑”菜单中的“撤消”进行重置。若要重置“默认命名空间”属性值, 必须在“项目设计器”中编辑该属性。</p>
属性	将属性的声明和用法更改为新名称。
类型	将类型的所有声明和所有用法都更改为新名称, 包括构造函数和析构函数。对于部分类型, 重命名操作将传播到其所有部分。

备注

对于实现/重写其他类型中成员的成员, 或者由其他类型中的成员实现/重写的成员, 重命名这些成员时, Visual Studio 会显示一个对话框, 说明该重命名操作将导致级联更新。如果单击“继续”, 则重构引擎将递归查找并重命名相应基类型和派生类型中的所有成员, 这里的基类型和派生类型是与要重命名的成员之间存在实现/重写关系的类型。

下面的代码示例包含具有实现/重写关系的成员。

C#

```
interface IBase
{
    void Method();
}
public class Base
{
    public void Method()
    {
    }
    public virtual void Method(int i)
    {
    }
}
public class Derived : Base, IBase
{
    public new void Method()
    {
    }
    public override void Method(int i)
    {
    }
}
public class C : IBase
{
    public void Method()
    {
    }
}
```

在上面的示例中, 重命名 C.Method() 也将重命名 Ibase.Method(), 这是因为 C.Method() 实现 Ibase.Method()。接着, 重构引擎通过递归查找发现 Ibase.Method() 是由 Derived.Method() 实现的, 因此将重命名 Derived.Method()。重构引擎不会重命名 Base.Method(), 因为 Derived.Method() 并不重写 Base.Method()。如果在“重命名”对话框没有选中“重命名重载”, 则重构引擎将到此停止。

如果选中了“重命名重载”, 则重构引擎将重命名 Derived.Method(int i) (因为它重载 Derived.Method())、Base.Method(int i) (因为它由 Derived.Method(int i) 重载) 和 Base.Method() (因为它是 Base.Method(int i) 的重载方法)。

 **注意**

当重命名在引用程序集中定义的成员时, 将显示一个对话框, 说明该重命名操作将导致生成错误。

请参见

任务

[如何: 重命名标识符](#)

概念

重构

“重命名”对话框

使用“重命名”可以对代码中表示符号(例如字段、局部变量、方法、命名空间、属性和类型)的标识符进行重命名。

新名称

指定标识符的新名称，该标识符是要重命名的代码元素。

位置

标识在执行重命名操作时要搜索的命名空间。

预览引用更改

指定修改代码之前在“预览更改 - 重命名”对话框中预览更改。

在注释中搜索

指定选择该复选框后在注释中搜索。

在字符串中搜索

指定选择该复选框后在字符串中搜索。

重命名重载

指定在重构操作中包括方法重载。如果选中该选项，则重构引擎将仅重命名该类型中的同名方法 - 不对基类型或继承类型中的方法进行重命名。

注意

此选项只可供方法使用。

备注

当 **Rename** 重构操作对注释和字符串进行搜索后，文本将根据全局搜索和替换操作中的简单字符串匹配进行更改。建议在选定“在注释中搜索”或“在字符串中搜索”的情况下选择“预览引用更改”。

重命名命名空间时，Visual Studio 还更新“项目设计器”的[“应用程序页”](#)上的“默认命名空间”属性。此属性不能通过选择“编辑”菜单中的“撤消”进行重置。若要重置“默认命名空间”属性值，必须在“项目设计器”中编辑该属性。

请参见

任务

[如何：重命名标识符](#)

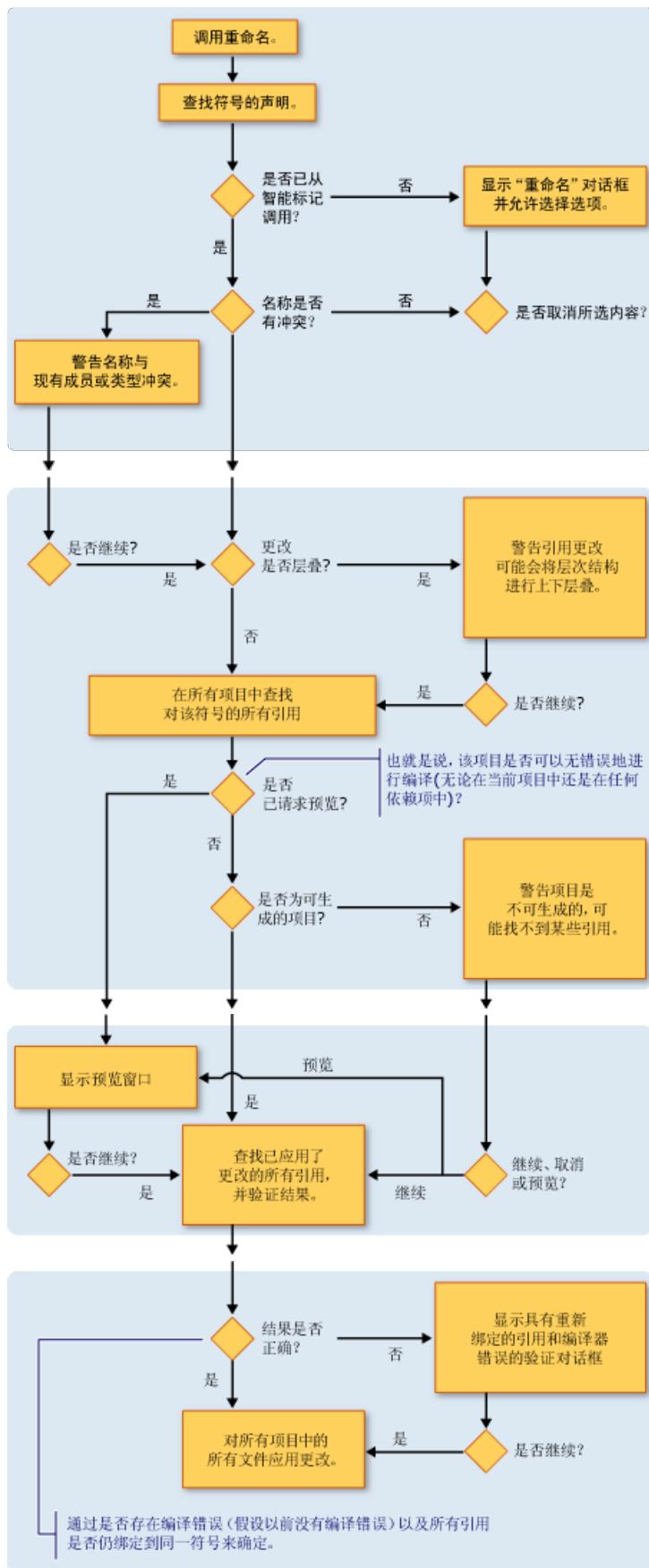
参考

[重命名](#)

[“预览更改”对话框](#)

重命名重构的流程图

下面的关系图阐释了完成重命名重构操作的步骤序列。



请参见
参考
[重命名](#)
[概念](#)

重构

如何：重命名标识符

下面的过程描述如何重命名代码中的标识符。使用此过程可以执行[重命名](#)重构操作。

重命名标识符

- 按照下面的示例部分中的描述，创建控制台应用程序。

有关更多信息，请参见[控制台应用程序](#)。

- 将光标放在方法声明或方法调用中的 MethodB 上。

- 从“重构”菜单中选择“重命名”。随即显示“重命名”对话框。

您还可以键入键盘快捷键 F2 来显示“重命名”对话框。

还可以右击光标，指向上下文菜单中的“重构”，然后单击“重命名”以显示“重命名”对话框。

- 在“新名称”字段中键入 **MethodC**。

- 选择“在注释中搜索”复选框。

- 单击“确定”。

- 在“预览更改”对话框中单击“应用”。

使用智能标记重命名标识符

- 按照下面的示例部分中的描述，创建控制台应用程序。

有关更多信息，请参见[控制台应用程序](#)。

- 在 MethodB 的声明中，键入方法标识符或在方法标识符上按 Backspace。此标识符下将显示智能标记提示。

注意

只能在标识符声明中使用智能标记来调用重命名重构功能。

- 键入键盘快捷键 Shift+Alt+F10，然后按向下键以显示智能标记菜单。

- 或 -

将鼠标指针移至智能标记提示上，以显示该智能标记。然后将鼠标指针移至该智能标记上，并单击向下箭头以显示智能标记菜单。

- 要调用不具有代码更改预览的重命名重构功能，请选择“将‘<identifier1>’重命名为‘<identifier2>’”菜单项。对“<identifier1>”的所有引用都将自动更新为“<identifier2>”。

- 或 -

要调用具有代码更改预览的重命名重构功能，请选择“带预览重命名”菜单项。将会显示“预览更改”对话框。

示例

若要设置此示例，请创建一个名为 `RenameIdentifier` 的控制台应用程序，然后使用以下代码替换 `Class1`。有关更多信息，请参见[控制台应用程序](#)。

```
class ProtoClassA
{
    // Invoke on 'MethodB'.
    public void MethodB(int i, bool b) { }
}

class ProtoClassC
{
    void D()
{
```

```
ProtoClassA MyClassA = new ProtoClassA();

// Invoke on 'MethodB'.
MyClassA.MethodB(0, false);
}
```

[请参见](#)

[参考](#)

[重命名](#)

[概念](#)

[重构](#)

封装字段

利用“封装字段”重构操作，可以从现有字段快速创建属性，然后使用对新属性的引用无缝更新代码。

当某个字段为 [public\(C# 参考\)](#) 时，其他对象可以直接访问该字段并对其进行修改，而不会被拥有该字段的对象检测到。通过使用 [属性\(C# 编程指南\)](#) 封装该字段，可以禁止对字段的直接访问。

为了创建新属性，“封装字段”操作将为要封装为 [private\(C# 参考\)](#) 的字段更改访问修饰符，然后为该字段生成 **get** 和 **set** 访问器。在某些情况下，将只生成 **get** 访问器，例如，当字段声明为只读时。

重构引擎将用“[封装字段](#)”对话框的“更新引用”部分所指定区域中对新属性的引用更新代码。

备注

仅当将光标与字段声明置于同一行时，才可以执行“封装字段”操作。

对于声明多个字段的声明，“封装字段”使用逗号作为字段之间的边界，并对与光标位于同一行且距离光标最近的某个字段启动重构。也可以通过在声明中选择该字段的名称来指定要封装的字段。

此重构操作生成的代码将由“[封装字段代码段](#)”来建模。代码段是可修改的。有关更多信息，请参见[如何：管理代码段](#)。

有关何时使用字段以及何时使用属性的更多信息，请参见[属性过程与字段](#)。

请参见

任务

[如何：用封装字段重构代码](#)

概念

重构

[代码段 \(C#\)](#)

“封装字段”对话框

使用此对话框可指定[封装字段](#)重构操作的设置。

字段名

标识生成新属性的字段的当前名称。

属性名

指定“封装字段”生成的新属性的名称。重构操作会自动生成一个唯一的属性名。不过，您可以将此名称更改为任何有效的标识符。

注意

如果您输入的名称是一个无效的标识符或与现有名称冲突，则将出现错误，而且重构将无法继续。

更新引用:

指定重构引擎是否使用对新属性的引用自动更新代码。

选项	说明
External	指定将封闭类型外对该字段的每个引用都替换为对新属性的引用。在封闭类型内，该字段的使用保持不变。
全部	<p>指定将对该字段的每个引用都替换为对新属性的引用。</p> <p> 注意</p> <p>“封装字段”将不会更新构造函数内的字段引用。</p>

预览引用更改

指定在修改代码之前将在“预览更改 - 封装字段”对话框中显示对代码所做的更改。

在注释中搜索

指定重构引擎将在代码注释中搜索要更新的现有字段的引用。

在字符串中搜索

指定重构引擎将在字符串值中搜索要更新的现有字段的引用。

备注

当“封装字段”重构操作对注释和字符串进行搜索后，文本将根据全局搜索和替换操作中的简单字符串匹配进行更改。若要避免出现错误，请在选定“在注释中搜索”或“在字符串中搜索”的情况下选择“预览引用更改”。

请参见

任务

[如何：用封装字段重构代码](#)

参考

[“预览更改”对话框](#)

如何：用封装字段重构代码

下面的过程描述如何从现有字段创建属性，然后通过引用新属性来更新代码。使用此过程可以执行[封装字段](#)重构操作。

从字段创建属性

- 按下面示例中所述创建控制台应用程序。

有关更多信息，请参见[控制台应用程序模板](#)。

- 在[代码和文本编辑器](#)中，将光标放在声明中要封装的字段的名称上。在下面的示例中，将光标放在单词 width 上：

```
public int width, height;
```

- 从“重构”菜单中选择“封装字段”。随即显示“[封装字段](#)”对话框。

也可以通过键入键盘快捷键 Ctrl+R、Ctrl+F 来显示“[封装字段](#)”对话框。

还可以右击光标，指向上下文菜单上的“重构”，然后单击“[封装字段](#)”来显示“[封装字段](#)”对话框。

- 指定设置。
- 按 Enter，或单击“确定”按钮。
- 如果选择了“预览引用更改”选项，则将打开“[预览引用更改](#)”窗口。单击“应用”按钮。

源文件中将显示下面的 **get** 和 **set** 访问器代码：

```
public int Width
{
    get
    {
        return width;
    }

    set
    {
        width = value;
    }
}
```

Main 方法中的代码也将更新为新的 Width 属性名。

```
Square mySquare = new Square();
mySquare.Width = 110;
mySquare.height = 150;
// Output values for width and height.
Console.WriteLine("width = {0}", mySquare.Width);
```

示例

若要设置此示例，请创建一个名为 EncapsulateFieldExample 的控制台应用程序，然后使用以下代码替换 Class1。有关更多信息，请参见[控制台应用程序](#)。

```
class Square
{
    // Select the word 'width' then use Encapsulate Field.
    public int width, height;
}

class MainClass
```

```
{  
    public static void Main()  
    {  
        Square mySquare = new Square();  
        mySquare.width = 110;  
        mySquare.height = 150;  
        // Output values for width and height.  
        Console.WriteLine("width = {0}", mySquare.width);  
        Console.WriteLine("height = {0}", mySquare.height);  
    }  
}
```

[请参见](#)

[参考](#)

[封装字段](#)

[概念](#)

[重构](#)

提取接口

“提取接口”是一项重构操作，提供了一种使用来自现有类、结构或接口的成员创建新接口的简单方法。

当几个客户端使用类、结构或接口中成员的同一子集时，或者当多个类、结构或接口具有通用的成员子集时，在接口中嵌入成员子集将很有用。有关使用接口的更多信息，请参见[接口 \(C# 编程指南\)](#)。

“提取接口”在新文件中生成接口，并将光标定位于新文件的开头。使用[“提取接口”对话框](#)，可以指定要提取到新接口中的成员、新接口的名称以及所生成的文件的名称。

备注

仅当将光标定位于包含要提取成员的类、结构或接口中时，才可以访问此功能。当光标处于此位置时，调用“提取接口”重构操作。

在类或结构中调用“提取接口”时，将修改基和接口列表，以包括新接口名称。而在接口中调用“提取接口”时，将不修改基和接口列表。

请参见

任务

[如何：用“提取接口”重构代码](#)

概念

[重构](#)

“提取接口”对话框

使用此对话框可为[提取接口](#)重构操作指定设置。可通过“重构”菜单获得此对话框。

新接口名

指定要生成的接口的名称。

生成的名称

显示被添加到从中提取接口的类型(类或结构)的基和接口列表中的名称。

新文件名

指定将包含新接口的新文件的名称。

选择构成接口的公共成员

列出可用于填充新接口的所有有效成员。包括方法、索引器、属性和事件。

选择要提取到新接口中的每个成员。可以使用“全选”或“取消全选”成员右侧的按钮来构成接口。

请参见

任务

[如何: 用“提取接口”重构代码](#)

概念

[重构](#)

如何：用“提取接口”重构代码

使用此过程可以执行[提取接口](#)重构操作。

使用“提取接口”

- 按下面示例中所述创建控制台应用程序。

有关更多信息，请参见[控制台应用程序](#)。

- 将光标置于 MethodB 中后，选择“重构”菜单中的“提取接口”。随即显示“[提取接口](#)”对话框。

您还可以键入键盘快捷键 Ctrl+R、Ctrl+I 来显示“提取接口”对话框。

也可以右击光标，指向上下文菜单中的“重构”，然后单击“提取接口”以显示“提取接口”对话框。

- 单击“全选”。

- 单击“确定”按钮。

您将看到新文件 IProtoA.cs 和下面的代码：

```
using System;
namespace TopThreeRefactorings
{
    interface IProtoA
    {
        void MethodB(string s);
    }
}
```

示例

若要设置此示例，请创建一个名为 ExtractInterface 的控制台应用程序，然后使用以下代码替换 Class1。有关更多信息，请参见[控制台应用程序](#)。

```
// Invoke Extract Interface on ProtoA.
// Note: the extracted interface will be created in a new file.
class ProtoA
{
    public void MethodB(string s) { }
```

请参见

参考

[提取接口](#)

概念

[重构](#)

将局部变量提升为参数

"将局部变量提升为参数"是一项 Visual C# 重构操作，可提供一种简单的方法，以在正确更新调用站点的同时将变量从局部使用移动至方法、索引器或构造函数参数。

要执行"将局部变量提升为参数"操作，首先要将光标置于希望提升的变量上。声明该变量的语句还必须为变量赋值或赋予表达式。定位光标后，通过键入键盘快捷键或从快捷菜单中选择相应的命令，可调用"将局部变量提升为参数"操作。

调用"将局部变量提升为参数"操作时，变量将被添加到成员参数列表的结尾处。对已修改成员的所有调用都将使用新参数（将替代最初赋给该变量的表达式）立即进行更新，并保留代码，以使其像变量提升之前那样正常工作。有关更多信息，请参见[如何：将局部变量提升为参数](#)。

备注

将常数值赋值给提升的变量时，此重构操作效果最好。必须声明并初始化该变量，而不能仅声明或仅赋值。

请参见

任务

[如何：将局部变量提升为参数](#)

概念

[重构](#)

如何：将局部变量提升为参数

使用此过程可以执行“将局部变量提升为参数”重构操作。有关更多信息，请参见[将局部变量提升为参数](#)。

将局部变量提升为参数

1. 创建一个控制台应用程序，并按照下面的示例中所述对其进行设置。有关更多信息，请参见[控制台应用程序模板](#)。
2. 将光标放在 MethodB 中的 i 上。
3. 从“重构”菜单中选择“将局部变量提升为参数”。

也可以通过键入键盘快捷键 Ctrl+R、Ctrl+P 来完成重构操作。

还可以右击光标，指向上下文菜单上的“重构”，然后单击“将局部变量提升为参数”来完成重构操作。

现在，MethodB 应具有参数 int i，而调用 ProtoA.MethodB 将把 0 作为值进行传递。

示例

若要设置此示例，请创建一个名为 PromoteLocal 的控制台应用程序，然后使用以下代码替换 Class1。有关更多信息，请参见[控制台应用程序模板](#)。

```
class ProtoA
{
    public static void MethodB()
    {
        // Invoke on 'i'
        int i = 0;
    }
}

class ProtoC
{
    void MethodD()
    {
        ProtoA.MethodB();
    }
}
```

请参见

[概念](#)

[重构](#)

移除参数

Remove Parameters 是一项重构操作，提供了从方法、索引器或委托中移除参数的简单方法。**Remove Parameters** 将更改声明；在调用成员的任何位置，都会将参数移除以反映新声明。

要执行 **Remove Parameters** 操作，请首先将光标放在方法、索引器或委托上。当光标处于合适位置时，可以通过以下方式来调用“移除参数”操作：从“重构”菜单中选择该操作、键入键盘快捷键或从上下文菜单中选择相应命令。

调用“移除参数”命令时，将显示“移除参数”对话框。有关更多信息，请参见[“移除参数”对话框](#)或[如何：移除参数](#)。

备注

可以从方法声明或方法调用中移除参数。请将光标置于方法声明或委托名称中，然后调用“移除参数”。

► 警告

使用“移除参数”可以移除在成员体中引用的参数，但不会移除在方法体中对该参数的引用。这可能会在您的代码中引入生成错误。但是，您可以使用[“预览更改”对话框](#)在执行重构操作之前检查代码。

如果正在移除的参数在调用方法的过程中被修改，则移除参数的同时也将移除所做的修改。例如，如果重构操作将某个方法调用由：

```
MyMethod(param1++, param2);
```

更改为

```
MyMethod(param2);
```

`param1` 将不会递增。

请参见

任务

[如何：移除参数](#)

概念

[重构](#)

“移除参数”对话框

使用此对话框可标识要在执行[移除参数](#)重构操作过程中移除的参数。

参数

允许您从列表中移除参数。

预览方法签名

显示移除了参数后的新方法签名。

预览引用更改

指定在选择了该复选框后显示“预览更改 - 移除参数”对话框。

备注

如果正在移除的参数在调用方法的过程中被修改，则移除参数的同时也将移除所做的修改。例如，如果重构操作将某个方法调用由：

```
MyMethod(param1++, param2);
```

更改为：

```
MyMethod(param2);
```

param1 将不会递增。建议预览引用更改。

请参见

[任务](#)

[如何：移除参数](#)

[参考](#)

[移除参数](#)

[“预览更改”对话框](#)

如何 : 移除参数

使用此过程可执行 **Remove Parameters** 重构操作。有关更多信息, 请参见[移除参数](#)。

移除参数

1. 创建一个控制台应用程序, 并设置下面的示例。

有关更多信息, 请参见[控制台应用程序模板](#)。

2. 在方法声明或方法调用中, 将光标放在 `A` 方法上。

3. 从“重构”菜单中选择“移除参数”以显示“移除参数”对话框。

也可以通过键入键盘快捷键 `Ctrl+R, Ctrl+V` 来显示“移除参数”对话框。

还可以右击光标, 指向上下文菜单上的“重构”, 然后单击“移除参数”来显示“移除参数”对话框。

4. 使用“参数”字段, 将光标放在 `int i` 上, 然后单击“移除”按钮。

5. 单击“确定”。

6. 在“预览更改 - 移除参数”对话框中单击“应用”。

示例

若要设置此示例, 请创建一个名为 `RemoveParameters` 的控制台应用程序, 然后使用以下代码替换 `Class1`。有关更多信息, 请参见[控制台应用程序模板](#)。

```
class A
{
    // Invoke on 'A'.
    public A(string s, int i) { }

    class B
    {
        void C()
        {
            // Invoke on 'A'.
            A a = new A("a", 2);
        }
    }
}
```

请参见

参考

[移除参数](#)

概念

[重构](#)

重新排列参数

“重新排列参数”是一项 Visual C# 重构操作，提供了一种对方法、索引器和委托的参数顺序进行更改的简单方法。“重新排列参数”会更改声明，并在调用该成员的所有位置重新排列参数，从而反映新的顺序。

通过先将光标置于方法、索引器或委托上，可以执行“重新排列参数”操作。定位光标后，通过键入键盘快捷键或从上下文菜单中选择相应的命令来调用“重新排列参数”操作。

调用“重新排列参数”时，将显示“重新排列参数”对话框。有关更多信息，请参见[“重新排列参数”对话框](#)和[如何：重新排列参数](#)。

备注

可以通过方法声明或方法调用来重新排列参数。要将光标置于方法声明或委托声明中，而不是置于正文中。

请参见

任务

[如何：重新排列参数](#)

概念

重构

“重新排列参数”对话框

使用此对话框可指定[重新排列参数](#)重构操作的参数顺序。

参数

允许您使用箭头按钮在列表中上移或下移参数。

预览方法签名

显示重新排列参数后方法签名的复本。

预览引用更改

指定在选择复选框时显示“预览更改 - 重新排列参数”对话框。

请参见

任务

[如何 : 重新排列参数](#)

参考

[重新排列参数](#)

[“预览更改”对话框](#)

如何：重新排列参数

可以为方法、索引器、构造函数和委托更改参数的顺序，并使用[重新排列参数](#)重构操作自动更新它们的调用站点。

重新排列参数

1. 创建一个[类库](#)并按照以下示例部分所述设置该类库。
2. 将光标放在方法声明或方法调用中的 MethodB 上。
3. 在“重构”菜单上单击“重新排列参数”。

- 或 -

键入键盘快捷键 Ctrl+R、Ctrl+O 以显示“重新排列参数”对话框。

- 或 -

右击光标，指向上下文菜单上的“重构”，然后单击“重新排列参数”以显示“重新排列参数”对话框。

将显示[“重新排列参数”对话框](#)。

4. 在“重新排列参数”对话框中，在“参数”列表中选择 int i。然后单击下移按钮。

- 或 -

拖动“参数”列表中 bool b 下的 int i。

5. 在“重新排列参数”对话框中单击“确定”。

如果在“重新排列参数”对话框中选择了“预览引用更改”选项，则将显示[“预览更改 - 重新排列参数”对话框](#)。它提供了签名和方法调用中 MethodB 的参数列表中的更改预览。

- a. 如果显示“预览更改 - 重新排列参数”对话框，则请单击“应用”。

在此示例中，更新了 MethodB 的方法声明和所有方法调用站点。

示例

若要设置此示例，请创建一个名为 ReorderParameters 的类库，然后使用以下代码替换 Class1。

```
class ProtoClassA
{
    // Invoke on 'MethodB'.
    public void MethodB(int i, bool b) { }
}

class ProtoClassC
{
    void D()
    {
        ProtoClassA MyClassA = new ProtoClassA();

        // Invoke on 'MethodB'.
        MyClassA.MethodB(0, false);
    }
}
```

请参见

[参考](#)

[重新排列参数](#)

[概念](#)

[重构](#)

“预览更改”对话框

利用“预览更改”对话框，可以在执行重构操作之前查看重构操作将对代码执行的所有引用更改。

预览引用更改是一个可选的重构过程。若要预览引用更改，请选中以下对话框中可用的“预览引用更改”选项：

- “重命名”对话框
- “封装字段”对话框
- “移除参数”对话框
- “重新排列参数”对话框

注意

通过从 Windows 窗体设计器的“属性网格”中执行重命名重构，可以直接修改代码。该操作不会显示“重命名”对话框或“预览更改”对话框。有关更多信息，请参见[重命名](#)重构。

<Refactor> <Code> 为 <RefactoredCode>

显示项目中的代码将发生更改的位置。代码语句显示为源文件节点下的节点。选择代码语句节点时，对应的引用显示为重构后的状态并突出显示在“预览代码更改”文本框中。

预览代码更改

显示程序，其中包括通过重构操作并入程序中的所有引用更改。

请参见

任务

- [如何：重命名标识符](#)
- [如何：用封装字段重构代码](#)
- [如何：移除参数](#)
- [如何：重新排列参数](#)

“重构警告”对话框

此警告对话框指示编译器没有完全理解程序，重构引擎可能没有更新所有合适的引用。此警告对话框还为您提供了在提交更改之前在[“预览更改”对话框](#)中预览代码的机会。

注意

如果方法中包含语法错误(IDE 以红色波浪下划线指示该错误)，则重构引擎将不会更新该方法中对元素的任何引用。下面的示例阐释了此行为。

默认情况下，如果执行重构操作而不预览引用更改，并且在程序中检测到编译错误，则开发环境将显示此警告对话框。

如果执行已启用“预览引用更改”的重构操作，并在程序中检测到编译错误，则开发环境将在“预览更改”对话框的底部显示下面的警告消息，以替代“重构警告”对话框：

当前未生成项目或其中一个依赖项。可能未更新引用。

此重构警告只可用于提供“预览引用更改”选项的重构操作，该选项在下列重构对话框中可用：

- [“重命名”对话框](#)
- [“封装字段”对话框](#)
- [“移除参数”对话框](#)
- [“重新排列参数”对话框](#)

每次显示此对话框

默认情况下此选项处于选定状态。如果选择此选项，则当在执行重构操作的过程中检测到编译错误时，将继续显示“重构警告”对话框。

清除此复选框将对以后的重构操作禁用此警告对话框。如果要清除此复选框，然后再为将来的重构操作重新启用此警告对话框，则请在[“选项”对话框 -> “文本编辑器”->“C#/J#”->“高级”](#)中选择“如果在重构时存在生成错误则发出警告”选项。

继续

继续当前的重构操作，而不预览引用更改。

预览

打开[“预览更改”对话框](#)，以便可以预览代码。

取消

取消当前的重构操作。代码中将不发生任何更改。

示例

下面的代码示例阐释重构引擎不更新引用的情况。如果使用重构将 **example** 重命名为其他名称，则 **ContainsSyntaxError** 中的引用将不会更新，而其他两个引用将被更新。

```
public class Class1
{
    static int example;

    static void ContainsSyntaxError()
    {
        example = 20
    }

    static void ContainsSemanticError()
    {
        example = "Three";
    }

    static void ContainsNoError()
    {
```

```
        example = 1;  
    }  
}
```

[请参见](#)

[概念](#)

[重构](#)

“验证结果”对话框

当重构引擎在重构验证过程中检测到编译错误或重新绑定问题时，便会显示此对话框。

验证结果

标识包含因重构操作而引起编译错误或重新绑定问题的语句。

预览问题

显示在程序中引入编译错误或重新绑定问题的引用更改(或重构的代码)的预览。

备注

在以下情况下，不会显示此对话框：

- 启动重构操作之前代码生成无法解析的编译错误，并且没有因重构操作而引入任何重新绑定问题。
- 您可以在“预览更改”对话框中清除所有引用。
- 当您在警告对话框中看到指示存在命名冲突并且重构操作将跳过验证过程的提示时，请单击“是”。

重新绑定问题

如果重构操作由于疏忽而导致代码引用绑定到其他对象，而不是绑定到最初绑定到的对象，则会出现重新绑定问题。“验证结果”对话框区分两种重新绑定问题的差异。

其定义将不再是重命名的符号的引用

如果引用指的不再是已重命名的符号，则会出现这种重新绑定问题。例如，考虑以下代码：

```
class Example
{
    private int a;
    public Example(int b)
    {
        a = b;
    }
}
```

如果使用重构功能将 `a` 重命名为 `b`，则将显示此对话框。现在，对重命名变量 `a` 的引用绑定到传递到构造函数的参数，而不是绑定到字段。

其定义现在将成为重命名的符号的引用

如果以前指的并不是重命名的符号的引用现在指的是重命名的符号，则将出现这种重新绑定问题。例如，考虑以下代码：

```
class Example
{
    private static void Method(object a)
    {
    }
    private static void OtherMethod(int a)
    {
    }
    static void Main(string[] args)
    {
        Method(5);
    }
}
```

如果您使用重构功能将 `OtherMethod` 重命名为 `Method`，则将显示此对话框。现在，`Main` 中的引用指的是接受 `int` 参数的重载方法，而不是接受 `object` 参数的重载方法。

请参见 参考

[“预览更改”对话框](#)

[概念](#)

[重构](#)

代码段 (C#)

Visual Studio 提供了一项称为代码段的新功能。可以使用代码段键入短的别名，然后在通用编程构造中将其展开。例如，**for** 代码段创建一个空的 **for** 循环。有些代码段为外侧代码段，这些代码段允许您先选择代码行，然后选择要并入选定代码行的代码段。例如，选择代码行，然后激活 **for** 代码段，便可以创建一个 **for** 循环，选定的这些代码行在该循环块内。代码段可以使程序代码的编写更快、更容易、更可靠。

使用代码段

在“代码编辑器”中使用代码段的常用方法为：先键入别名的简称（即代码段快捷方式），然后按 Tab 键。IntelliSense 菜单还提供了一个“插入代码段”菜单命令，一个可插入“代码编辑器”中的代码段的列表。键入 Ctrl+K，再键入 X，便可以激活该代码段列表。有关更多信息，请参见[如何：使用代码段 \(C#\)](#) 和 [如何：使用外侧代码段](#)。

一旦选定某个代码段，该代码段的文本就会自动插入光标所在位置。此时，代码段中的任何可编辑字段都将突出显示为黄色，并自动选择第一个可编辑字段。当前选定的字段在红色框中。例如，在 **for** 代码段中，可编辑字段是初始值设定项变量（默认情况下为 `i`）和长度表达式（默认情况下为 `length`）。

选定某字段后，用户可以为该字段键入新值。按 Tab 可以循环通过代码段的可编辑字段；按 Shift+Tab 可以按相反的顺序循环通过这些字段。单击某字段可将光标置于该字段中，双击某字段可选择该字段。突出显示某字段时，可能会显示工具提示，以提供该字段的说明。

只有给定字段的第一个实例是可编辑的；突出显示该字段时，该字段的其他实例均概要显示。当您更改某个可编辑字段的值以后，在代码段中凡是用到该字段的地方都会对该字段进行更改。

按 Enter 或 Esc 将取消字段编辑，使“代码编辑器”恢复普通模式。

通过修改“选项”对话框“字体和颜色”窗格中的“代码段字段”设置，可以更改可编辑代码段字段的默认颜色。有关更多信息，请参见[如何：更改编辑器中使用的字体及其大小和颜色](#)。

创建代码段

除了默认情况下 Visual Studio 包括的代码段以外，还可以创建和使用自定义代码段。有关创建自定义代码段的更多信息，请参见[创建代码段](#)。

注意

对于 C# 代码段，用于指定 [<快捷方式>](#) 字段的有效字符包括：字母数字字符、数字符号 (#)、波形符 (~)、下划线字符 (_) 和短划线字符 (-)。

有关默认情况下 Visual C# 包括的代码段的更多信息，请参见[默认代码段](#)。

请参见

参考

[代码段选择器](#)

默认代码段

代码段插入器可在光标位置插入代码段，或在当前选定的代码周围插入外侧代码段。调用代码段插入器的方法有：通过“IntelliSense”菜单上的“插入代码段”或“外侧代码”命令；分别使用键盘快捷键 Ctrl+K 和 X 以及 Ctrl+K 和 S。

代码段插入器显示所有可用代码段的名称。代码段插入器中还包括一个输入对话框，可在此键入代码段的名称或代码段名称的一部分。代码段插入器将突出显示与代码段名称最接近的匹配项。按 Tab 键可随时消除代码段插入器，并插入当前选定的代码段。键入 Esc 或在“代码编辑器”中单击鼠标将消除代码段插入器，但不插入代码段。

默认代码段

默认情况下，Visual Studio 中包括下列代码段。

名称(或快捷方式)	说明	插入代码段的有效位置
#if	创建 <code>#if</code> 指令和 <code>#endif</code> 指令。	任意位置。
#region	创建 <code>#region</code> 指令和 <code>#endregion</code> 指令。	任意位置。
~	为包含类创建析构函数。	在类中。
attribute	为从 <code>Attribute</code> 派生的类创建声明。	在命名空间(包括全局命名空间)、类或结构中。
checked	创建 <code>checked</code> 块。	在方法、索引器、属性访问器或事件访问器中。
class	创建类声明。	在命名空间(包括全局命名空间)、类或结构中。
ctor	为包含类创建构造函数。	在类中。
cw	创建对 <code>WriteLine</code> 的调用。	在方法、索引器、属性访问器或事件访问器中。
do	创建 <code>do while</code> 循环。	在方法、索引器、属性访问器或事件访问器中。
else	创建 <code>else</code> 块。	在方法、索引器、属性访问器或事件访问器中。
enum	创建 <code>enum</code> 声明。	在命名空间(包括全局命名空间)、类或结构中。
equals	创建一个方法声明，该声明对 <code>Object</code> 类中定义的 <code>Equals</code> 方法进行重写。	在类或结构中。
exception	为某个从异常(默认情况下为 <code>Exception</code>)派生的类创建声明。	在命名空间(包括全局命名空间)、类或结构中。
for	创建 <code>for</code> 循环。	在方法、索引器、属性访问器或事件访问器中。
foreach	创建 <code>foreach</code> 循环。	在方法、索引器、属性访问器或事件访问器中。

forr	创建一个 <code>for</code> 循环，在每次循环之后递减循环变量。	在方法、索引器、属性访问器或事件访问器中。
if	创建 <code>if</code> 块。	在方法、索引器、属性访问器或事件访问器中。
索引器 (indexer)	创建索引器声明。	在类或结构中。
interface	创建 <code>interface</code> 声明。	在命名空间(包括全局命名空间)、类或结构中。
invoke	创建可安全调用事件的块。	在方法、索引器、属性访问器或事件访问器中。
iterator	创建迭代器。	在类或结构中。
iterindex	使用嵌套类创建“命名的”迭代器和索引器对。	在类或结构中。
lock	创建 <code>lock</code> 块。	在方法、索引器、属性访问器或事件访问器中。
mbox	创建对 <code>System.Windows.Forms.MessageBox.Show</code> 的调用。您可能需要添加对 <code>System.Windows.Forms.dll</code> 的引用。	在方法、索引器、属性访问器或事件访问器中。
namespace	创建 <code>namespace</code> 声明。	在命名空间(包括全局命名空间)中。
prop	创建属性声明和支持字段。	在类或结构中。
propg	创建只具有“get”访问器和支持字段的属性声明。	在类或结构中。
sim	创建 <code>staticint Main</code> 方法声明。	在类或结构中。
struct	创建 <code>struct</code> 声明。	在命名空间(包括全局命名空间)、类或结构中。
svm	创建 <code>staticvoid Main</code> 方法声明。	在类或结构中。
switch	创建 <code>switch</code> 块。	在方法、索引器、属性访问器或事件访问器中。
try	创建 <code>try-catch</code> 块。	在方法、索引器、属性访问器或事件访问器中。
tryf	创建 <code>try-finally</code> 块。	在方法、索引器、属性访问器或事件访问器中。
unchecked	创建 <code>unchecked</code> 块。	在方法、索引器、属性访问器或事件访问器中。
unsafe	创建 <code>unsafe</code> 块。	在方法、索引器、属性访问器或事件访问器中。

using	创建 <code>using</code> 指令。	在命名空间(包括全局命名空间)中。
while	创建 <code>while</code> 循环。	在方法、索引器、属性访问器或事件访问器中。

备注

快捷方式使 IntelliSense 可以自动将代码段填入“代码编辑器”，而不需要使用菜单。有关更多信息，请参见[如何 : 使用代码段 \(C#\)](#)。

请参见

任务

[如何 : 使用外侧代码段](#)

参考

[代码段选择器](#)

概念

[代码段 \(C#\)](#)

如何 : 使用代码段 (C#)

以下过程描述如何使用代码段。可以通过五种方式使用代码段：键盘快捷键，IntelliSense 自动完成，IntelliSense 完成单词列表，“编辑”菜单，上下文菜单。

通过键盘快捷键使用代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，将光标置于要插入代码段的位置。
3. 键入 Ctrl+K、Ctrl+X。
4. 从代码段插入器中选择代码段，然后按 Tab 或 Enter。

也可以键入代码段的名称，然后按 Tab 或 Enter。

通过 IntelliSense 自动完成使用代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，将光标置于要插入代码段的位置。
3. 为要添加到代码中的代码段键入快捷方式。
4. 键入两次 Tab 以调用代码段。

通过 IntelliSense 完成单词列表使用代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，将光标置于要插入代码段的位置。
3. 首先为要添加到代码中的代码段键入快捷方式。如果已打开自动完成，则将显示 IntelliSense 完成单词列表。如果该列表未显示，则按 Ctrl+Space 激活它。
4. 从完成单词列表中选择代码段。
5. 键入两次 Tab 以调用代码段。

通过“编辑”菜单使用代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，将光标置于要插入代码段的位置。
3. 从“编辑”菜单中选择“IntelliSense”，然后选择“插入代码段”命令。
4. 从代码段插入器中选择代码段，然后按 Tab 或 Enter。

也可以键入代码段的名称，然后按 Tab 或 Enter。

通过上下文菜单使用代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，将光标置于要插入代码段的位置。
3. 右击光标，然后从上下文菜单中选择“插入代码段”命令。
4. 从代码段插入器中选择代码段，然后按 Tab 或 Enter。

也可以键入代码段的名称，然后按 Tab 或 Enter。

请参见

任务

[如何 : 使用外侧代码段](#)

参考

[默认代码段](#)

[代码段选择器](#)

[概念](#)

[代码段 \(C#\)](#)

如何：使用外侧代码段

以下过程描述如何使用外侧代码段。可通过三种方法使用外侧代码段：通过键盘快捷键、通过“编辑”菜单，以及通过上下文菜单。

通过键盘快捷键使用外侧代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，选择要添加外侧代码段的文本。
3. 键入 Ctrl+K、Ctrl+S。
4. 使用鼠标，或者通过键入代码段的名称并按 Tab 或 Enter，从代码段列表中选择代码段。

通过“编辑”菜单使用外侧代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，选择要添加外侧代码段的文本。
3. 从“编辑”菜单中选择“IntelliSense”，然后选择“外侧代码”命令。
4. 从代码段插入器中选择代码段，然后按 Tab 或 Enter。

也可以键入代码段的名称，然后按 Tab 或 Enter。

通过上下文菜单使用外侧代码段

1. 在 Visual Studio IDE 中打开要编辑的文件。
2. 在“代码编辑器”中，选择要添加外侧代码段的文本。
3. 右击选定的文本，然后从上下文菜单中选择“外侧代码”命令。
4. 从代码段插入器中选择代码段，然后按 Tab 或 Enter。

也可以键入代码段的名称，然后按 Tab 或 Enter。

请参见

任务

[如何：使用代码段 \(C#\)](#)

参考

[默认代码段](#)

[代码段选择器](#)

概念

[代码段 \(C#\)](#)

如何：还原 C# 重构代码段

C# 重构操作依赖于以下目录中的代码段：

Installation directory\Microsoft Visual Studio 8\VC#\Snippets\language ID\Refactoring

如果此重构目录或此目录中的任何文件被删除或损坏，则 C# 重构操作在 IDE 中可能不起作用。以下过程可帮助您还原 C# 重构代码段。

可以通过代码段管理器验证 C# 重构代码段

1. 在“工具”菜单中，选择“代码段管理器”。
2. 在“代码段管理器”对话框中，从“语言”下拉列表中选择“Visual C#”。

“重构”文件夹应出现在树视图文件夹列表中。

在代码段管理器中还原重构代码段

1. 在“工具”菜单中，选择“代码段管理器”。
2. 在“代码段管理器”对话框中，从“语言”下拉列表中选择“Visual C#”。
3. 单击“添加”。此时出现“代码段目录”对话框，该对话框可帮助您定位和指定要添加回代码段管理器的目录。
4. 定位目录路径如下的“Refactoring”文件夹：

Installation directory\Microsoft Visual Studio 8\VC#\Snippets\language ID\Refactoring

5. 在“代码段目录”对话框中单击“打开”，然后在代码段管理器中单击“确定”。

修复重构代码段目录

1. 在“代码段管理器”对话框中，单击“联机搜索”。
2. 输入重构，然后单击“搜索”。

搜索结果应包括一个使您能够下载 .vsi 文件的网站，您可以使用该文件重新安装重构文件夹。

请参见

参考

[代码段管理器](#)

概念

[重构](#)

代码着色

代码编辑器会对标记和代码构造进行分析，以便可以方便地识别它们，并且可以将它们与代码编辑器中的其他代码内容区分开来。代码编辑器对代码进行了分析后，便会相应地对代码构造进行着色。

标记

代码编辑器将对以下标记类型进行着色。

- 注释
- 排除的代码
- 标识符
- 关键字
- 数字
- 运算符
- 预处理器关键字
- 字符串
- 字符串(C# @ 逐字字符串)
- 用户类型
- 用户类型(值类型)
- 用户类型(枚举)
- 用户类型(委托)
- XML CData 节
- XML 文档属性
- XML 文档注释
- XML 文档标记

可以使用“选项”对话框 ->“环境”->“字体和颜色”来修改默认的着色设置。

上下文关键字

代码编辑器会相应地对上下文关键字进行着色。在下面的示例中，类型 **yield** 着色后为青色，而关键字 **yield** 着色后为蓝色。



大括号匹配着色

代码编辑器有利于进行粗体着色或大括号匹配突出显示着色。

粗体着色

编辑下面的任何代码构造对时，字符串或代码构造对以粗体形式简要显示，指示它们之间的关联：

" "	字符串
@" "	原义字符串
#if, #endif	条件节的预处理器指令
#region, #endregion	条件节的预处理器指令
case, break	控制语句关键字
default, break	控制语句关键字
for, break	计算表达式关键字
for, continue	计算表达式关键字
foreach, break	计算表达式关键字
foreach, continue	计算表达式关键字
while, break	计算表达式关键字
while, continue	计算表达式关键字

通过取消选择“选项”对话框 ->“文本编辑器”->“常规”中的“自动突出显示分隔符”属性，可以禁用此功能。

突出显示着色

将光标定位到紧靠起始分隔符之前或紧靠结束分隔符之后时，将显示灰色的矩形，突出显示起始和结束分隔符，从而指示它们之间的关联。此功能可用于以下匹配对：

{}	大括号
[]	方括号
()	圆括号

示例

若要演示大括号匹配着色，请在代码编辑器中键入（不要复制和粘贴）以下代码。

```
class A
{
    public A()
    {
        if(true)
            int x =0;
        else
            int x =1;
    }
}
```

着色设置

通过[Visual Studio 设置](#)保存着色设置。

[请参见](#)

[参考](#)

[自动括号匹配](#)

作为源代码的元数据

作为源代码的元数据使您可以在只读缓冲区中查看显示为 C# 源代码的元数据。这样，就能够查看类型和成员的声明（但没有实现）。对于其源代码从您的项目或解决方案中不可用的类型或成员，您可以通过对它们运行“转到定义”命令来查看作为源代码的元数据。

注意

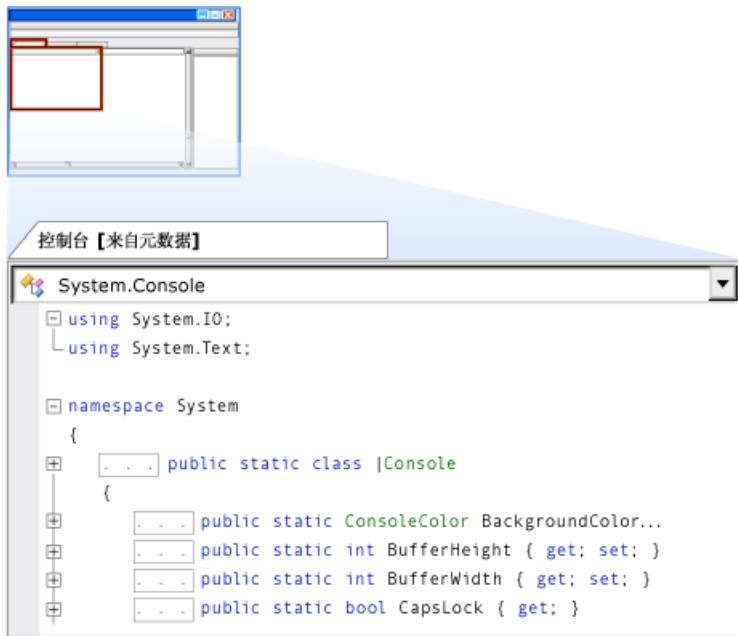
当试图对标记为内部的类型或成员运行“转到定义”命令时，集成开发环境（IDE）不将它们的元数据显示为源代码，无论引用的程序集是否为友元。

您可以在“代码编辑器”或“代码定义”窗口中查看作为源代码的元数据。

在“代码编辑器”中查看作为源代码的元数据

当对其源代码不可用的项运行“转到定义”命令时，包含该项的元数据（显示为源代码）的视图的选项卡式文档出现在“代码编辑器”中。类型的名称出现在文档的选项卡上，后面跟 “[从元数据]”。

例如，如果对 **Console** 运行“转到定义”命令，**Console** 的元数据作为 C# 源代码（看起来像它的声明，但没有实现）出现在“代码编辑器”中。



在“代码定义”窗口中查看作为源代码的元数据

当“代码定义”窗口活动或可见时，IDE 自动对“代码编辑器”中光标下的项以及在“类视图”或“对象浏览器”中选择的项执行“转到定义”命令。如果源代码对该项不可用，则 IDE 在“代码定义”窗口中显示该项的作为源代码的元数据。

例如，如果在“代码编辑器”中将光标放在单词 **Console** 中，则 **Console** 的元数据作为源代码显示在“代码定义”窗口中。此源代码有点类似于 **Console** 声明，但没有实现。

如果希望看到在“代码定义”窗口中出现的项的声明，必须显式使用“转到定义”命令，因为“代码定义”窗口的深度只有一级。

请参见

参考

[代码定义窗口](#)

[“查找符号结果”窗口](#)

Visual C# 键盘快捷键

Visual C# 提供了一些键盘快捷键，您可以使用这些快捷键(而无需使用鼠标或菜单)执行操作。

本节内容

[快捷键](#)

[常规开发设置](#)[默认快捷键](#)

[请参见](#)

[其他资源](#)

[Visual C#](#)

[使用 Visual C# IDE](#)

Visual C# IDE 设置

Visual C# 设置是工具窗口、菜单和键盘快捷键的预定义配置。这些设置是 [Visual Studio 设置](#) 功能的一部分，可以对后者进行自定义以适合您的工作习惯。

窗口和视图

功能	是否默认显示？	说明
类视图	否	<ul style="list-style-type: none"> “类视图”在“视图”菜单中可用。 启用了筛选功能。
“命令”窗口	否	
“动态帮助”窗口	否	<p>按 F1 键不会显示“动态帮助”窗口。</p> <p>有关“动态帮助”窗口的更多信息，请参见 如何：自定义动态帮助 或 如何：控制动态帮助窗口。</p>
对象浏览器	否	<ul style="list-style-type: none"> 默认情况下不显示继承成员。
“输出”窗口	否	
解决方案资源管理器	是	“解决方案资源管理器”显示并停靠在 IDE 的右侧。
起始页	是, 启动 IDE 时	“起始页”显示来自 Visual C# 的 MSDN RSS 源的文章。
任务列表 (Visual Studio)	否	
工具箱	是, 创建 Windows 窗体应用程序时	“工具箱”显示为一个折叠的窗口, 停靠在 IDE 的左侧。

键盘

功能	行为
快捷键	<p>Visual C# 支持以下快捷键设置：</p> <ul style="list-style-type: none"> Visual C# 2005 默认快捷键 Brief 默认快捷键 Emacs 默认快捷键 Visual C++ 2.0 默认快捷键 Visual Studio 6.0 默认快捷键

请参见

其他资源

[Visual C#](#)

[使用 Visual C# IDE](#)

Visual C# 2005 默认快捷键

集成开发环境 (IDE) 提供了多个预定义的键盘绑定方案。若要切换到“Visual C# 2005”键盘映射方案，请单击“工具”菜单上的“选项”，展开“环境”，再单击“键盘”。

此外，当在“导入和导出设置向导”中选中“Visual C# 开发设置”时，“Visual C# 2005”键盘映射方案为默认键盘映射方案。有关更多信息，请参见[如何：更改选择设置](#)。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

下列信息介绍了可用于“Visual C# 2005”键盘映射方案的默认键组合。

- [Visual C# 2005 方案 -> 全局快捷键](#)
- [Visual C# 2005 方案 -> HTML 设计器快捷键](#)
- [Visual C# 2005 方案 -> XML 设计器快捷键](#)
- [Visual C# 2005 方案 -> 控件操作快捷键](#)
- [Visual C# 2005 方案 -> 调试快捷键](#)
- [Visual C# 2005 方案 -> 搜索和替换快捷键](#)
- [Visual C# 2005 方案 -> 数据快捷键](#)
- [Visual C# 2005 方案 -> 文本导航快捷键](#)
- [Visual C# 2005 方案 -> 文本选择快捷键](#)
- [Visual C# 2005 方案 -> 文本操作快捷键](#)
- [Visual C# 2005 方案 -> 窗口管理快捷键](#)
- [Visual C# 2005 方案 -> 集成帮助快捷键](#)
- [Visual C# 2005 方案 -> 对象浏览器快捷键](#)
- [Visual C# 2005 方案 -> 宏快捷键](#)
- [Visual C# 2005 方案 -> 工具窗口快捷键](#)
- [Visual C# 2005 方案 -> 项目快捷键](#)
- [Visual C# 2005 方案 -> 图像编辑器快捷键](#)
- [Visual C# 2005 方案 -> 对话框编辑器快捷键](#)
- [Visual C# 2005 方案 -> 重构快捷键](#)
- [Visual C# 2005 方案 -> 托管资源编辑器快捷键](#)
- [Visual C# 2005 方案 -> 代码段快捷键](#)
- [Visual C# 2005 方案 -> 类关系图快捷键](#)
- [Visual C# 2005 方案 -> 书签窗口快捷键](#)
- [Visual C# 2005 方案 -> 快捷键和字符串编辑器快捷键](#)

请参见

任务

[如何：使用快捷组合键](#)

其他资源

[快捷键](#)

Visual C# 2005 方案 -> 全局快捷键

下列快捷组合键可用于集成开发环境 (IDE) 中的不同位置。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
关系图.属性	Alt + Enter	将焦点从关系图切换到“属性”窗口。
编辑.复制	Ctrl + C	将选定项复制到剪贴板。
编辑.剪切	Ctrl + X	从文件中删除选定项，并将其复制到剪贴板。
编辑.循环应用剪贴板中的复制项	Ctrl + Shift + V	将某个项从“剪贴板循环”粘贴到文件中的插入点，并自动选择所粘贴的项。可通过重复按这些快捷键来查看“剪贴板循环”上的每个项。
编辑.删除	Delete	删除插入点右侧的一个字符。
编辑.打开文件	Ctrl + Shift + G	显示“打开文件”对话框，在其中可以选择要打开的文件。
编辑.粘贴	Ctrl + V	在插入点插入剪贴板的内容。
编辑.重做	CTRL + Y	还原上次撤消的操作。
编辑.取消	Ctrl + Z	撤消上一编辑操作。
文件.打印	Ctrl + P	显示“打印”对话框，可在其中选择打印机设置。
文件.全部保存	Ctrl + Shift + S	保存当前解决方案中的所有文档和外部文件项目中的所有文件。
文件.保存选定项	Ctrl + S	保存当前项目中的选定项。
工具.转到命令行	CTRL + /	将指针放置在“标准”工具栏上的“查找/命令”框中。
视图.后退	Alt + 向右键	显示查看历史记录中的上一页。仅在“Web 浏览器”窗口中可用。
视图.编辑标签	F2	允许更改“解决方案资源管理器”中选定项的名称。
视图.前进	Alt + 向左键	显示查看历史记录中的下一页。仅在“Web 浏览器”窗口中可用。
视图.查看代码	F7	在编辑器的“代码”视图中显示选定项。
视图.视图设计器	Shift + F7	在“设计器”视图中显示选定项。

请参见

概念

Visual C# 2005 方案 -> HTML 设计器快捷键

只有在 HTML 设计器中修改文件时才能使用下列快捷键组合。某些组合键仅在该设计器的特定视图中可用。可在 HTML 设计器中使用的其他组合键包

括常规开发设置 -> 文本导航快捷键、常规开发设置 -> 文本选择快捷键和常规开发设置 -> 文本操作快捷键。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
格式.粗体	Ctrl + B	在粗体和正常之间切换选定文本。仅在“设计”视图中可用。
格式.转换为超链接	CTRL + L	选定文本后，显示“超链接”对话框。仅在“设计”视图中可用。
格式.插入书签	Ctrl + Shift + L	显示“书签”对话框。仅在“设计”视图中可用。
格式.斜体	Ctrl + I	在斜体和正常之间切换选定文本。仅在“设计”视图中可用。
格式.下划线	Ctrl + U	在下划线和正常之间切换选定文本。仅在“设计”视图中可用。
布局.在其左侧插入列	Ctrl + Alt + 向左键	将一列添加到表中当前列的左边。仅在“设计”视图中可用。
布局.在其右侧插入列	Ctrl + Alt + 向右键	将一列添加到表中当前列的右边。仅在“设计”视图中可用。
布局.在其上方插入行	Ctrl + Alt + 向上键	在表中的当前行之上添加一行。仅在“设计”视图中可用。
布局.在其下方插入行	Ctrl + Alt + 向下键	在表中的当前行之下添加一行。仅在“设计”视图中可用。
项目.添加内容页	Ctrl + M, CtrI + C	向网站添加一个新的 *.aspx 文件，并在 HTML 设计器中打开该文件。仅在“设计”视图中可用。
视图.自动关闭标记重写	Ctrl + Shift + 句点	临时重写当前标记的默认关闭标记行为。有关更多信息，请参见 标记专用选项 。仅在“源”视图中可用。
视图.详细信息	Ctrl + Shift + Q	显示没有可视化表示的 HTML 元素(如：注释、脚本和绝对定位元素的定位点)的图标。仅在“设计”视图中可用。
视图.编辑主表	Ctrl + M, CtrI + M	在“源”视图中打开 *.master 文件。仅在“设计”视图中可用。
视图.下一个视图	Ctrl + Page Down	在当前文档的“设计”视图、“源”视图和“服务器代码”视图之间切换。在所有视图中可用。
视图.非可视控件	Ctrl + Alt + Q	显示非图形元素的符号，如 div、span、form 和 script 元素。仅在“设计”视图中可用。

视图.显示智能标记	Shift + Alt + F10	显示 Web 服务器控件常用命令的智能标记菜单。仅在“设计”视图中可用。
视图.视图设计器	Shift + F7	切换到当前文档的“设计”视图。仅在“源”视图中可用。
视图.查看标记	Shift + F7	切换到当前文档的“源”视图。仅在“设计”视图中可用。
视图.可视边框	Ctrl + Q	在支持设置为零的 BORDER 属性的 HTML 元素周围显示一个 1 像素边框。此类 HTML 元素的示例包括表、表的单元格和区域。仅在“设计”视图中可用。
窗口.上一选项卡	Ctrl + Page Up	在当前文档的“设计”视图、“源”视图和“服务器代码”视图之间切换。在所有视图中可用。

请参见

[参考](#)

[HTML 设计器](#)

[概念](#)

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> XML 设计器快捷键

使用 XML 设计器时可以使用下列快捷键组合。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
架构.折叠	Ctrl + 减号 (-)	折叠嵌套元素。仅在 XML 设计器的“架构”视图中可用。
架构.展开	Ctrl + 等号 (=)	展开嵌套元素。仅在 XML 设计器的“架构”视图中可用。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 控件操作快捷键

下列快捷组合键可用于在设计图面上移动和选择控件以及更改控件的大小。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.下移控件	向下键	在设计图面上，将选定的控件以 1 像素为增量向下移动。
编辑.将控件移动到下侧网格	Ctrl + 向下键	在设计图面上，将选定的控件以 8 像素为增量向下移动。
编辑.左移控件	向左键	在设计图面上，将控件以 1 像素为增量向左移动。
编辑.将控件移动到左侧网格	Ctrl + 向左键	在设计图面上，将控件以 8 像素为增量向左移动。
编辑.右移控件	向右键	在设计图面上，将控件以 1 像素为增量向右移动。
编辑.将控件移动到右侧网格	Ctrl + 向右键	在设计图面上，将控件以 8 像素为增量向右移动。
编辑.上移控件	向上键	在设计图面上，将控件以 1 像素为增量向上移动。
编辑.将控件移动到上侧网格	Ctrl + 向上键	在设计图面上，将控件以 8 像素为增量向上移动。
编辑.选择下一个控件	Tab	根据控件的 TabIndex 属性移动到页面上的下一个控件。
编辑.选择上一个控件	Shift + Tab	移回到页面上选定的前一个控件。
编辑.显示平铺网格	Enter	在设计图面上显示网格。
编辑.向下调大控件大小	Shift + 向下键	在设计图面上，以 1 像素为增量，增加控件的高度。
编辑.将控件调大到下侧网格	Ctrl + Shift + 向下键	在设计图面上，以 8 像素为增量，增加控件的高度。
编辑.向左调整控件大小	Shift + 向右键	在设计图面上，以 1 像素为增量，减少控件的宽度。
编辑.将控件调大到左侧网格	Ctrl + Shift + 向左键	在设计图面上，以 8 像素为增量，减少控件的宽度。
编辑.向右调整控件大小	Shift + 向右键	在设计图面上，以 1 像素为增量，增加控件的宽度。
编辑.将控件调大到右侧网格	Ctrl + Shift + 向右键	在设计图面上，以 8 像素为增量，增加控件的宽度。
编辑.向上调整控件大小	Shift + 向上键	在设计图面上，以 1 像素为增量，减少控件的高度。
编辑.将控件调大到上侧网格	Ctrl + Shift + 向上键	在设计图面上，以 8 像素为增量，减少控件的高度。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 调试快捷键

下列快捷键组合可在调试代码时使用。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
调试.应用代码更改	Alt + F10	启动生成操作，利用它可以通过“编辑并继续”功能应用对正在调试的代码所作的更改。
调试.自动窗口	Ctrl + D, Ctrl + A	显示“自动”窗口，以查看在当前过程中，目前所执行中的变量的值。
调试.全部中断	Ctrl + Alt + Break	临时停止执行调试会话中的所有进程。仅可用于“运行”模式。
调试.在函数处中断	Ctrl + D, Ctrl + N	显示“新断点”对话框。
调试.断点	Ctrl + D, Ctrl + B	显示“断点”对话框，可以在其中添加和修改断点。
调试.调用堆栈	Ctrl + D, Ctrl + C	显示“调用堆栈”窗口，以显示当前执行线程的所有活动过程或堆栈帧的列表。仅可用于“运行”模式。
调试.删除所有断点	Ctrl + Shift + F9	清除项目中的所有断点。
调试.反汇编	Ctrl + Alt + D	显示“反汇编”窗口。
调试.启用断点	Ctrl + F9	将断点从禁用切换到启用。
调试.异常	Ctrl + D, Ctrl + E	显示“异常”对话框。
调试.即时	Ctrl + D, Ctrl + I	显示“即时”窗口，在此可计算表达式和执行个别命令。
调试.局部变量	Ctrl + D, Ctrl + L	显示“局部变量”窗口，以查看当前堆栈帧中每个过程的变量及变量值。
调试.内存1	Ctrl + Alt + M, 1	显示“内存 1”窗口，以查看大的缓冲区、字符串以及无法在“监视”或“变量”窗口中清楚显示的其他数据。
调试.内存2	Ctrl + Alt + M, 2	显示“内存 2”窗口，以查看大的缓冲区、字符串以及无法在“监视”或“变量”窗口中清楚显示的其他数据。
调试.内存3	Ctrl + Alt + M, 3	显示“内存 3”窗口，以查看大的缓冲区、字符串以及无法在“监视”或“变量”窗口中清楚显示的其他数据。

调试.内存4	Ctrl + Alt + M, 4	显示“内存4”窗口，以查看大的缓冲区、字符串以及无法在“监视”或“变量”窗口中清楚显示的其他数据。
调试.模块	Ctrl + D, Ctrl + M	显示“模块”窗口，利用它可以查看程序使用的.dll 或.exe 文件。在多进程调试中，可以右击再单击“显示所有程序的模块”。
调试.进程	Ctrl + D, Ctrl + P	显示“进程”窗口。可用于“运行”模式。
调试.快速监视	Ctrl + D, Ctrl + Q	显示包含选定表达式的当前值的“快速监视”对话框。仅适用于“中断”模式。使用该命令可检查变量、属性或尚未定义监视表达式的其他表达式的当前值。
调试.寄存器	Ctrl + D, Ctrl + R	显示“寄存器”窗口，此窗口显示用于调试本机代码应用程序的寄存器内容。
调试.重新启动	Ctrl + Shift + F5	结束调试会话，重新生成并从头开始运行应用程序。可用于“中断”模式和“运行”模式。
调试.运行到光标处	Ctrl + F10	在“中断”模式下，从当前语句继续执行代码，直到选定语句。“当前执行行”边距指示符出现在“边距指示符”栏中。在“设计”模式下，启动调试器并执行代码(执行到光标所在的位置)。
调试.脚本资源管理器	Ctrl + Alt + N	显示“脚本资源管理器”窗口，该窗口列出了正在调试的文档集。可用于“运行”模式。
调试.设置下一语句	Ctrl + Shift + F10	在选择的代码行上设置执行点。
调试.显示下一语句	Alt + 数字键区中的 *	突出显示要执行的下一条语句。
调试.启动	F5	自动附加调试器，并从“<项目> 属性”对话框中指定的启动项目运行应用程序。如果为“中断”模式，则更改为“继续”。
调试.开始执行不调试	Ctrl + F5	在不调用调试器的情况下运行代码。
调试.逐语句	F11	在执行进入函数调用后，逐条语句执行代码。
调试.进入并单步执行当前进程	Ctrl + Alt + F11	可从“进程”窗口使用。
调试.跳出	Shift + F11	执行当前执行点所在函数的剩余行。
调试.跳出当前进程	Ctrl + Shift + Alt + F11	可从“进程”窗口使用。
调试.逐过程	F10	执行下一行代码，但不执行任何函数调用。
调试.逐过程执行当前进程	Ctrl + Alt + F10	可从“进程”窗口使用。
调试.停止调试	Shift + F5	停止运行程序中的当前应用程序。可用于“中断”模式和“运行”模式。
调试.线程	Ctrl + D, Ctrl + T	显示“线程”窗口，以查看当前进程的所有线程以及它们的相关信息。

调试.切换断点	F9	在当前行设置或移除断点。
调试.切换反汇编	Ctrl + D, Ctrl + I	显示当前源文件的反汇编信息。仅适用于“中断”模式。
调试.监视	Ctrl + Alt + W, 1	显示“监视 1”窗口，以查看所选变量或监视表达式的值。
调试.监视2	Ctrl + Alt + W, 2	显示“监视 2”窗口，以查看所选变量或监视表达式的值。
调试.监视3	Ctrl + Alt + W, 3	显示“监视 3”窗口，以查看所选变量或监视表达式的值。
调试.监视4	Ctrl + Alt + W, 4	显示“监视 4”窗口，以查看所选变量或监视表达式的值。
调试器上下文菜单. 断点窗口.删除	Alt + F9, D	移除选定断点。仅在“断点”窗口中可用。
调试器上下文菜单. 断点窗口.转到反汇编	Alt + F9, A	显示“反汇编”窗口。仅在“断点”窗口中可用。
调试器上下文菜单. 断点窗口.转到源代码	Alt + F9, S	转到代码文件中选定断点的位置。仅在“断点”窗口中可用。
工具.附加到进程	Ctrl + Alt + P	显示“附加到进程”对话框，利用它可以在单个解决方案中同时调试多个程序。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 数据快捷键

下列快捷键组合可用于在集成开发环境 (IDE) 中处理数据。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
数据.列	CTRL + L	将新列添加到数据集底部。仅在“数据集编辑器”中可用。
数据.执行	Ctrl + Alt + F5	运行当前活动的数据库对象。
数据.插入列	Insert	在数据集中选定列的上方插入新列。仅在“数据集编辑器”中可用。
数据.运行选定内容	Ctrl + Q	运行 SQL 编辑器中的当前选择。
数据.显示数据源	Shift + Alt + D	显示“数据源”窗口。
数据.逐语句	Alt + F5	使当前的活动数据库对象按步骤进入调试模式。
查询设计器.取消检索数据	Ctrl + T	取消或停止当前运行的查询。仅在“查询和视图设计器”中可用。
查询设计器.条件	Ctrl + 2	显示“查询和视图设计器”的“条件”窗格。仅在“查询和视图设计器”中可用。
查询设计器.关系图	Ctrl + 1	显示“查询和视图设计器”的“关系图”窗格。仅在“查询和视图设计器”中可用。
查询设计器.执行SQL	CTRL + R	执行查询。仅在“查询和视图设计器”中可用。
查询设计器.转到行	CTRL + G	在“结果”窗格中时，会将焦点移动到停靠在设计器底部的工具条上。仅在“查询和视图设计器”中可用。
查询设计器.联接模式	Ctrl + Shift + J	启用 JOIN 模式。仅在“查询和视图设计器”中可用。
查询设计器.结果	Ctrl + 4	显示“查询和视图设计器”的“结果”窗格。仅在“查询和视图设计器”中可用。
查询设计器.SQL	Ctrl + 3	显示“查询和视图设计器”的“SQL”窗格。仅在“查询和视图设计器”中可用。
视图.数据集	Ctrl + Alt + D	显示报表设计器的“报表数据集”窗口。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 文本导航快捷键

在文本编辑器中使用下列快捷键组合可以在打开文档中的任何位置执行移动。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.左移字符	向左键	将插入点向左移动一个字符。
编辑.右移字符	向右键	将插入点向右移动一个字符。
编辑.清除书签	Ctrl + B, Ctrl + C	移除当前文档中所有未命名的书签。
编辑.文档结尾	Ctrl + End	将插入点移动到文档的最后一行。
编辑.文档开始	Ctrl + Home	将插入点移动到文档首行。
编辑.转到	CTRL + G	显示“转到行”对话框。
编辑.转到大括号	Ctrl +]	将插入点移动到文档中的下一个大括号处。
编辑.向下移动一行	向下键	将插入点下移一行。
编辑.行尾	End	将插入点移动到当前行的行尾。
编辑.行首	Home	将插入点移动到行首。
编辑.向上移动一行	向上键	将插入点上移一行。
编辑.下一书签	Ctrl + B, Ctrl + N	将插入点移动到下一个书签所在位置。
编辑.下一错误	Ctrl + Shift + F12	移至“错误列表”窗口中的下一个错误项，此窗口将自动滚动到编辑器中受影响的文本部分。
编辑.向下翻页	Page Down	在编辑器窗口中向下滚动一屏。
编辑.向上翻页	Page Up	在编辑器窗口中向上滚动一屏。
编辑.上一书签	Ctrl + B, Ctrl + P	将插入点移动到上一个书签所在位置。
编辑.快速信息	Ctrl + K, Ctrl + I	基于当前语言显示 快速信息 。
编辑.向下滚动一行	Ctrl + 向下键	将文本向下滚动一行。仅在文本编辑器中可用。
编辑.向上滚动一行	Ctrl + 向上键	将文本向上滚动一行。仅在文本编辑器中可用。
编辑.切换书签	Ctrl + K, Ctrl + K - 或 - Ctrl + B, Ctrl + T	在当前行处设置或移除书签。

编辑.视图底部	Ctrl + Page Down	移动到活动窗口的最后一个可见行。
编辑.视图顶部	Ctrl + Page Up	移动到活动窗口的第一个可见行。
编辑.下一字	Ctrl + 向右键	将插入点向右移动一个单词。
编辑.上一字	Ctrl + 向左键	将插入点向左移动一个单词。
视图.浏览下一个	Ctrl + Shift + 1	移动到某一项的下一个定义、声明或引用。在“对象浏览器”和“类视图”窗口中可用。
视图.浏览上一个	Ctrl + Shift + 2	移动到某一项的上一个定义、声明或引用。在“对象浏览器”和“类视图”窗口中可用。
视图.向后定位	Ctrl + 减号 (-)	移动到已浏览的上一个代码行。
视图.向前定位	Ctrl + Shift + 减号 (-)	移动到已浏览的下一个代码行。
视图.弹出浏览上下文	Ctrl + Shift + 8	移动到当前文件中的代码调用的上一项。
视图.向前浏览上下文	Ctrl + Shift + 7	移动到当前文件中的代码调用的下一项。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 文本选择快捷键

可以在文本编辑器中使用下列快捷键组合选择打开文档中的文本。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.向左扩展一个字符	Shift + 向右键	将光标左移一个字符以扩展所选内容。
编辑.向左扩展一个字符列	Shift + Alt + 向左键	将光标左移一个字符以扩展列选择。
编辑.向右扩展一个字符	Shift + 向右键	将光标右移一个字符以扩展所选内容。
编辑.向右扩展一个字符列	Shift + Alt + 向右键	将光标右移一个字符以扩展列选择。
编辑.文档结尾扩展	Ctrl + Shift + End	选择从光标位置到文档最后一行的文本。
编辑.文档开始扩展	Ctrl + Shift + Home	选择从光标位置到文档第一行的文本。
编辑.扩展转到大括号	Ctrl + Shift +]	将光标移至下一个大括号处，扩展选定内容。
编辑.向下扩展一行	Shift + 向下键	从光标位置开始，将选定文本向下扩展一行。
编辑.向下扩展列	Shift + Alt + 向下键	将指针下移一行，以扩展列的选定内容。
编辑.扩展到行尾	Shift + End	选择从光标位置到当前行行尾的文本。
编辑.行尾扩展列	Shift + Alt + End	将光标移至行尾，扩展列选择。
编辑.扩展到行首	Shift + Home	选择从光标位置到行首的文本。
编辑.行首扩展列	Shift + Alt + Home	将光标移至行首，以扩展列选择。
编辑.向上扩展一行	Shift + 向上键	从光标位置开始，向上逐行选择文本。
编辑.向上扩展列	Shift + Alt + 向上键	将光标上移一行以扩展列的选定内容。
编辑.向下扩展一页	Shift + Page Down	将选定内容向下扩展一页。
编辑.向上扩展一页	Shift + Page Up	将选定内容向上扩展一页。
编辑.全选	Ctrl + A	选择当前文档中的所有内容。
编辑.选择当前字	Ctrl + Shift + W	选择包含光标的单词或光标右侧的单词。
编辑.选择到最后一个返回	Ctrl + 等号 (=)	选择从“编辑器”中的当前位置到上一位置之间的内容。
编辑.扩展到视图底部	Ctrl + Shift + Page Down	将光标移动到视图中的最后一行以扩展选定内容。
编辑.扩展到视图顶部	Ctrl + Shift + Page Up	将选定范围扩展到活动窗口的顶部。

编辑.扩展到下一字	Ctrl + Shift + 向右键	将选定内容向右扩展一个单词的位置。
编辑.向后扩展一个字列	Ctrl + Shift + Alt + 向右键	将光标右移一个单词以扩展列选择。
编辑.扩展到上一字	Ctrl + Shift + 向左键	将选定内容向左扩展一个单词的位置。
编辑.向前扩展一个字列	Ctrl + Shift + Alt + 向左键	将光标左移一个单词以扩展列选择。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 文本操作快捷键

可以在文本编辑器中使用下列快捷键组合在打开文档中删除、移动或者格式化文本。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.分行	Enter	<p>插入一个新行。</p> <p> 注意</p> <p>在某些编辑器中，例如“HTML 设计器”的“设计”视图，Enter 的行为取决于上下文。有关更多信息，请参见您所使用的编辑器的文档。</p>
编辑.字符转置	Ctrl + T	交换光标两侧的字符。例如，AC BD 成为 AB CD。仅在文本编辑器中可用。
编辑.折叠标记	Ctrl + M, Ctrl + T	隐藏选定的 HTML 标记，并在其位置显示省略号 (...)。可以通过将鼠标指针置于 ... 之上，将完整标记作为工具提示查看。
编辑.折叠到定义	Ctrl + M, Ctrl + O	自动确定在代码中创建区域的逻辑边界(如过程)，然后隐藏它们。
编辑.注释选定内容	Ctrl + E, Ctrl + C	使用编程语言的正确注释语法将当前行代码标记为注释。
编辑.完成单词	Ctrl + K, Ctrl + W	显示基于当前语言的“完整单词”。
编辑.复制参数提示	Ctrl + Shift + Alt + C	将 IntelliSense 显示的参数信息复制到剪贴板。
编辑.向后删除	Backspace	删除光标左侧的一个字符。
编辑.删除水平空白	Ctrl + E, Ctrl + \	折叠选定内容中的空白，如果没有选定内容，则删除与光标相邻的空白。
编辑.编排文档格式	Ctrl + E, Ctrl + D	按照“选项”对话框“文本编辑器”部分中语言的“格式设置”窗格上指定的设置，对语言应用缩进和空格格式设置。
编辑.格式化选定内容	Ctrl + E, Ctrl + F	根据周围的代码行，正确缩进选定的代码行。
编辑.生成方法存根	Ctrl + K, Ctrl + M	<p>为光标所停留方法调用创建新的方法声明。</p> <p>有关更多信息，请参见 生成方法存根 (Stub)。</p>
编辑.隐藏选定内容	Ctrl + M, Ctrl + H	隐藏选定文本。信号图标标记隐藏文本在文件中的位置。
编辑.插入制表符	Tab	使文本行缩进指定数目的空格。

编辑.插入代码段	Ctrl + K, Ctrl + X	插入代码段。 有关更多信息, 请参见 代码段 (C#) 。
编辑.剪切行	CTRL + L	将所有选定的行剪切到“剪贴板”; 如果尚未选定任何内容, 则将当前行剪切到“剪贴板”。
编辑.删除行	Ctrl + Shift + L	删除所有选定行; 如果没有选定行, 则删除当前行。
编辑.上开新行	Ctrl + Enter	在光标上方插入一个空行。
编辑.下开新行	Ctrl + Shift + Enter	在光标之下插入一个空行。
编辑.行转置	Shift + Alt + T	将包含光标的行移动到下一行的下方。
编辑.列出成员	Ctrl + J	修改代码时, 列出当前类的成员以完成语句。
编辑.转换为小写	Ctrl + U	将选定文本更改为小写字符。
编辑.转换为大写	Ctrl + Shift + U	将选定文本更改为大写字符。
编辑.改写模式	Insert	在插入和改写插入模式之间切换。仅在使用文本编辑器时可用。
编辑.参数信息	Ctrl + Shift + 空格键	基于当前语言显示包含当前参数信息的工具提示。仅在 HTML 设计器的“源”视图中可用。
编辑.粘贴参数提示	Ctrl + Shift + Alt + P	将线前从 IntelliSense 复制的参数信息粘贴到光标所指示位置。
编辑.停止隐藏当前区域	Ctrl + M, Ctrl + U	移除当前选定区域的大纲显示信息。
编辑.停止大纲显示	Ctrl + M, Ctrl + P	从整篇文档中移除所有大纲显示信息。
编辑.交换定位点	Ctrl + E, Ctrl + A	交换当前选定内容的定位点与结束点。
编辑.左缩进	Shift + Tab	将选定行左移一个制表位。
编辑.切换所有大纲显示	Ctrl + M, Ctrl + L	在隐藏和显示状态之间切换所有以前被标记为隐藏的文本部分。
编辑.切换大纲显示展开	Ctrl + M, Ctrl + M	在隐藏和显示状态之间切换当前选定的隐藏文本部分。
编辑.切换任务列表快捷方式	Ctrl + E, Ctrl + T	在当前行处设置或移除快捷方式。
编辑.切换自动换行	Ctrl + E, Ctrl + W	启用或禁用编辑器中的自动换行。

编辑.取消注释选定内容	Ctrl + E, Ctrl + U	从代码的当前行中移除注释语法。
编辑.查看空白	Ctrl + E, Ctrl + S - 或 - Ctrl + R, Ctrl + W	显示或隐藏空格和制表符标记。
编辑.字删除直至结尾处	Ctrl + Delete	删除光标右侧的单词。
编辑.字删除直至开始处	Ctrl + Backspace	删除光标左侧的单词。
编辑.字转置	Ctrl + Shift + T	对调光标两侧的单词。例如, End Sub 将更改为读取 Sub End 。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 窗口管理快捷键

下列快捷组合键可在工具和文档窗口中用于进行移动、关闭或导航。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
视图.全屏	Shift+ Alt + Enter	切换“全屏”模式的打开和关闭。
视图.向后定位	Ctrl + 减号 (-)	返回到导航历史记录中的上一个文档或窗口。
视图.向前定位	Ctrl + 等号 (=)	向前移动到导航历史记录中的下一个文档或窗口。
窗口.激活文档窗口	Esc	关闭菜单或对话框，取消正在进行的操作，或将焦点放在当前文档窗口中。
窗口.关闭文档窗口	Ctrl + F4	关闭当前 MDI 子窗口。
窗口.关闭工具窗口	Shift + Esc	关闭当前工具窗口。
窗口.移动到导航栏	Ctrl + F2	当编辑器处于“代码”视图或“服务器代码”视图中时，将指针移至位于代码编辑器顶部的下拉栏中。
窗口.下一个文档窗口	Ctrl + F6	逐个窗口地循环通过 MDI 子窗口。
窗口.下一个文档窗口 导航栏	Ctrl + Tab	显示 IDE 导航器，并选中第一个文档窗口。
窗口.下一窗格	Alt + F6	移动到下一工具窗口。
窗口.下一选项卡	Ctrl + Page Down	移动到文档或窗口中的下一个选项卡。
窗口.下一个工具窗口 导航栏	Alt + F7	显示 IDE 导航器，并选中第一个工具窗口。
窗口.上一个文档窗口	Ctrl + Shift + F6	移动到编辑器或设计器中的上一个文档。
窗口.上一个文档窗口 导航栏	Ctrl + Shift + TAB	显示 IDE 导航器，并选中上一个文档窗口。
窗口.上一窗格	Shift + Alt + F6	移动到上次选定的窗口。
窗口.上一个拆分窗格	Shift + F6	在文档拆分窗格视图中移动到上一个窗格。
窗口.上一选项卡	Ctrl + Page Up	移动到文档或窗口中的上一个选项卡。
窗口.上一个工具窗口 导航栏	Shift + Alt + F7	显示 IDE 导航器，并选中上一个工具窗口。

窗口.显示EZMDI文件列表	Ctrl + Alt + 向下键	显示仅列出所有打开文档的弹出窗口。
----------------	------------------	-------------------

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 集成帮助快捷键

下列快捷组合键可用于在“帮助”中的主题之间查看和移动。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
帮助.目录	CTRL + F1, CTRL + C	针对 MSDN 中包含的文档显示“目录”窗口。
帮助.动态帮助	Ctrl + F1, Ctrl + D	显示“动态帮助”窗口。
帮助.F1帮助	F1	显示与选定的当前用户界面相对应的帮助主题。
帮助.帮助收藏夹	Ctrl + F1, Ctrl + F	显示“帮助收藏夹”。
帮助.如何实现	Ctrl + F1, Ctrl + H	显示与选定用户设置对应的“如何实现”页。
帮助.索引	Ctrl + F1, I	针对 MSDN 中包含的文档显示“索引”窗口。
帮助.索引结果	Ctrl + F1, Ctrl + T	显示“索引结果”窗口。
帮助.下一个主题	Alt + 向下键 - 或 - Alt + 向右键	显示目录中的下一主题。仅在“帮助”(Web) 浏览器窗口中可用。
帮助.上一个主题	Alt + 向上键 或 Alt + 向右键	显示目录中的前一主题。仅在“帮助”(Web) 浏览器窗口中可用。
帮助.搜索	Ctrl + F1, Ctrl + S	显示其“搜索”选项卡为活动状态的 Visual Studio 帮助页。利用该页可以在 MSDN 中包含的文档中搜索单词或词组。
帮助.搜索结果	Ctrl + F1, Ctrl + R	显示 Visual Studio 帮助页，该页带有“搜索”选项卡，并且焦点列在最近执行的搜索所产生主题列表中。
帮助.窗口帮助	Shift + F1	显示与当前用户界面相对应的“帮助”主题。

请参见

概念

[Visual C# 2005 默认快捷键](#)

[Visual Studio 设置](#)

Visual C# 2005 方案 -> 对象浏览器快捷键

下列快捷键组合可在“对象浏览器”中使用。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.查找符号	Alt + F12	显示“查找和替换”对话框的“查找符号”窗格。
编辑.转到声明	Ctrl + F12	显示代码中选定符号的定义。
编辑.转到定义	F12	显示代码中选定符号的声明。
编辑.快速查找符 号	Shift + Alt + F1 2	在文件中搜索选定对象或成员，并在“查找符号结果”窗口中显示匹配项。
视图.对象浏览器	Ctrl + Alt + J	显示“对象浏览器”，以查看可用于包的类、属性、方法、事件和常数，以及项目中的对象库和过程。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 宏快捷键

使用宏时可使用下列快捷键组合。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
视图.宏资源管理器	Alt + F8	显示“宏资源管理器”，其中列出了可供当前解决方案使用的所有宏。
工具.宏 IDE	Alt + F11	启动宏 IDE，即 Visual Studio 宏。
工具.记录临时宏	Ctrl+ Shift + R	将 Visual Studio IDE 置于宏记录模式。
工具.运行临时宏	Ctrl + Shift + P	播放记录的宏。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 工具窗口快捷键

下列快捷键组合允许显示特定的工具窗口。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
工具.代码段管理器	Ctrl + K, Ctrl + B	显示 代码段管理器 ，可以利用它搜索代码段并将其插入文件中。
视图.书签窗口	Ctrl + W, Ctrl + B	显示“书签”窗口。
视图.类视图	Ctrl + W, Ctrl + C	显示“类视图”窗口。
视图.类视图转到搜索组合框	Ctrl + K, Ctrl + V	更改“类视图搜索”框中的焦点。
视图.代码定义窗口	Ctrl + W, Ctrl + D	显示“代码定义”窗口。
视图.命令窗口	Ctrl + W, Ctrl + A	显示“命令”窗口，利用它可以键入控制集成开发环境 (IDE) 的命令。
视图.文档大纲	Ctrl + W, Ctrl + U	显示“文档大纲”窗口以查看当前文档的平面或分层大纲。
视图.错误列表	Ctrl + W, Ctrl + E	显示“错误列表”窗口。
视图.查找符号结果	Ctrl + W, Ctrl + Q	
视图.对象浏览器	Ctrl + W, Ctrl + J	
视图.输出	Ctrl + W, Ctrl + O	显示“输出”窗口以查看运行时的状态消息。
视图.挂起签入	Ctrl + W, Ctrl + G	
视图.属性窗口	Ctrl + W, Ctrl + P	显示“属性”窗口，该窗口列出当前选定项的设计时属性和事件。
视图.属性页	Shift+ F4	显示当前选定项的属性页。
视图.资源视图	Ctrl + W, Ctrl + R	显示“资源视图”窗口。

视图.服务器资源管理器	Ctrl + W, Ctrl + L	显示“服务器资源管理器”，利用它可以查看并操作数据库服务器、事件日志、消息队列、Web 服务和其他操作系统服务。
视图.解决方案资源管理器	Ctrl + W, Ctrl + S	显示“解决方案资源管理器”，它列出当前解决方案中的项目和文件。
视图.任务列表	Ctrl + W, Ctrl + T	显示“任务列表”窗口，在其中可以对任务、注释、快捷方式、警告和错误信息进行自定义、分类和管理。
视图.工具箱	Ctrl + W, Ctrl + X	显示“工具箱”，它包含可包括在代码中或与代码一起使用的控件和其他项。
视图.Web浏览器	Ctrl + W, Ctrl + W	显示“Web 浏览器”窗口，利用它可以查看 Internet 上的网页。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 项目快捷键

下列快捷键组合可用于向项目添加新项、生成项目、打开文件或打开项目。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
生成.生成解决方案	F6	生成解决方案。
生成.取消	Ctrl + Break	停止当前生成过程。
生成.编译	Ctrl + F7	创建一个对象文件，其中包含选定文件的机器码、链接器指令、节、外部引用以及函数名/数据名。
文件.新建文件	Ctrl + N	显示“新建文件”对话框，在此可以选择要添加到当前项目中的新文件。
文件.新建项目	Ctrl + Shift + N	显示“新建项目”对话框。
文件.打开文件	Ctrl + O	显示“打开文件”对话框。
文件.打开项目	Ctrl + Shift + O	显示“打开项目”对话框，在其中可将现有项目添加到解决方案。
项目.添加类	Shift + Alt + C	显示“添加新项”对话框，将类模板选择为默认值。
项目.添加现有项	Shift + Alt + A	显示“添加现有项”对话框，利用它可以将现有文件添加到当前项目中。
项目.添加新项	Ctrl + Shift + A	显示“添加新项”对话框，利用它可以向当前项目添加新文件。
项目.重写	Ctrl + Alt + Insert	允许重写派生类中的基类方法。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 图像编辑器快捷键

下表包括默认情况下绑定到键的图像编辑器命令的快捷键。若要更改快捷键，请在“工具”菜单上单击“选项”，展开“环境”，再单击“键盘”。有关更多信息，请参见[如何：使用快捷组合键](#)。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

命令	键	说明
图像.喷枪工具	Ctrl + A	使用具有选定大小和颜色的喷枪绘制。
图像.画笔工具	Ctrl + B	使用具有选定形状、大小和颜色的画笔绘制。
图像.复制和大纲选择	Ctrl + Shift + U	创建当前选定内容的副本并绘制其轮廓。如果当前选定内容包含背景色，当您选择 透明 后，会排除背景色。
图像.不透明处理	Ctrl + J	使当前的所选内容成为 透明或不透明 。
图像.椭圆工具	Ctrl + P	使用选定的线宽和颜色绘制一个椭圆。
图像.橡皮工具	Ctrl + Shift + I	清除图像的一部分(使用当前的背景色)。
图像.填充椭圆工具	Ctrl + Shift + Alt + P	绘制一个实心椭圆。
图像.填充矩形工具	Ctrl + Shift + Alt + R	绘制一个实心矩形。
图像.填充圆角矩形工具	Ctrl + Shift + Alt + W	绘制一个实心圆角矩形。
图像.填充工具	Ctrl + F	填充一块区域。
图像.水平翻转	Ctrl + H	水平翻转图像或选定内容。
图像.垂直翻转	Shift + Alt + H	垂直翻转图像或选定内容。
图像.较大画笔	Ctrl + =	增加画笔大小(每个方向上增加一个像素)。若要减小画笔大小，请参见本表中的“图像.较小画笔”。
图像.直线工具	CTRL + L	用选定形状、大小和颜色绘制一条直线。
图像.放大工具	Ctrl + M	切换到“放大”工具，利用它可以放大图像的特定部分。
图像.放大	Ctrl + Shift + M	在当前放大倍数和 1:1 放大倍数之间切换。
图像.新建图像类型	Insert	打开 “新建 <设备> 图像类型”对话框 ，使用它可以创建其他图像类型的图像。
图像.下一种颜色	Ctrl +] - 或 - Ctrl + 向右键	将绘制前景色更改为下一个调色板颜色。

图像.下一适当颜色	Ctrl + Shift +] - 或 - Shift + Ctrl + 向右键	将绘制背景色更改为下一个调色板颜色。
图像.椭圆轮廓工具	Shift + Alt + P	用一个轮廓绘制实心椭圆。
图像.矩形轮廓工具	Shift + Alt + R	用一个轮廓绘制实心矩形
图像.圆角矩形轮廓工具	Shift + Alt + W	用一个轮廓绘制实心圆角矩形。
图像.铅笔工具	Ctrl + I	使用单像素铅笔绘制。
图像.前一种颜色	Ctrl + [- 或 - Ctrl + 向左键	将绘制前景色更改为以前的调色板颜色。
图像.前一适当颜色	Ctrl + Shift + [- 或 - Shift + Ctrl + 向左键	将绘制背景色更改为以前的调色板颜色。
图像.矩形选择工具	Shift + Alt + S	选择图像的一个矩形部分, 以执行移动、复制或编辑。
图像.矩形工具	Alt + R	使用选定的线宽和颜色绘制一个矩形。
图像.旋转90度	Ctrl + Shift + H	将图像或选定内容旋转 90 度。
图像.圆角矩形工具	Alt + W	使用选定的线宽和颜色绘制一个圆角矩形。
图像.显示网格	Ctrl + Alt + S	切换像素网格(选中或清除“ 网格设置 ”对话框中的“像素网格”选项)。
图像.显示平铺网格	Ctrl + Shift + Alt + S	切换平铺网格(选中或清除“ 网格设置 ”对话框中的“平铺网格”选项)。
图像.小画笔	Ctrl + .(句点)	将画笔大小减小到一个像素。(请参见本表中的 <code>Image.LargerBrush</code> 和 <code>Image.SmallerBrush</code> 。)
图像.较小画笔	Ctrl + -(减号)	减小画笔大小(每个方向上减小一个像素)。若要再次扩展画笔大小, 请参见本表中的 <code>Image.LargerBrush</code> 。
图像.文本工具	Ctrl + T	打开 “文本工具”对话框 。
图像.使用选择内容作为画笔	Ctrl + U	将当前所选内容用作画笔来绘制。
图像.放大	Ctrl + Shift + .(句点) - 或 - Ctrl + 向上键	增加当前视图的放大倍数。

图像.缩小	Ctrl + ,(逗号) - 或 - Ctrl + 向下键	减小当前视图的放大倍数。
-------	-------------------------------------	--------------

有关如何将资源添加到托管项目的信息，请参见《.NET Framework 开发人员指南》中的[应用程序中的资源](#)。有关手动将资源文件添加到托管项目、访问资源、显示静态资源以及将资源字符串分配给属性的信息，请参见[演练:本地化 Windows 窗体](#)和[演练:将本地化资源用于 ASP.NET](#)。

要求

无

[请参见](#)

[参考](#)

[图像编辑器](#)

[概念](#)

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 对话框编辑器快捷键

下表包括对话框编辑器命令的默认键盘快捷键。若要更改快捷键，请在“工具”菜单上单击“选项”，展开“环境”，再单击“键盘”。有关更多信息，请参见[如何：使用快捷组合键](#)。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

命令	键	说明
格式.底部对齐	Ctrl + Shift + 向下键	将选定控件的底边与主导控件对齐。
格式.居中对齐	Shift + F9	将选定控件的垂直中心与主导控件对齐。
格式.左对齐	Ctrl + Shift + 向左键	将选定控件的左边缘与主导控件对齐。
格式.中间对齐	F9	将选定控件的水平中心与主导控件对齐。
格式.右对齐	Ctrl + Shift + 向右键	将选定控件的右边缘与主导控件对齐。
格式.顶部对齐	Ctrl + Shift + 向上键	将选定控件的上边缘与主导控件对齐。
格式.按钮底	Ctrl + B	沿对话框底部的中心放置选定的按钮。
格式.按钮右	CTRL + R	将选定按钮放置在对话框的右上角。
格式.水平居中	Ctrl + Shift + F9	使控件在对话框内水平居中。
格式.垂直居中	Ctrl + F9	使控件在对话框内垂直居中
格式.检查助记键	Ctrl + M	检查助记键的唯一性。
格式.按内容调整大小	Shift + F7	调整选定控件的大小，使其适合标题文本。
格式.横向间隔	Alt + 向右键	让选定控件在水平方向均匀分布。
格式.纵向间隔	Alt + 向下键	让选定控件在垂直方向均匀分布。
格式.Tab键顺序	Ctrl + D	设置对话框中控件的顺序。
格式.测试对话框	Ctrl + T	运行对话框以测试外观和行为。
格式.切换辅助线	CTRL + G	编辑对话框时在无网格、参考线和网格之间循环。

有关如何将资源添加到托管项目的信息，请参见《.NET Framework 开发人员指南》中的[应用程序中的资源](#)。有关如何手动将资源文件添加到托管项目、访问资源、显示静态资源以及如何将资源字符串分配给属性的信息，请参见[演练：本地化 Windows 窗体](#)和[演练：将本地化资源用于 ASP.NET](#)。

请参见

参考

[对话框编辑器](#)

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 搜索和替换快捷键

下列快捷键组合可用于在一个文件或多个文件中搜索文本，或是搜索对象和成员。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.查找	Ctrl + F	显示“查找和替换”对话框的“快速查找”选项卡。
编辑.查找所有引用	Ctrl + K, Ctrl + R	显示所有符号引用的查找位置的列表。
编辑.在文件中查找	Ctrl + Shift + F	显示“查找和替换”对话框的“在文件中查找”选项卡。
编辑.查找下一个	F3	查找上次搜索文本的下一个匹配项。
编辑.查找下一个选定项	Ctrl + F3	在文档中查找当前选定文本的下一个匹配项。
编辑.查找上一个	Shift + F3	查找搜索文本的上一个匹配项。
编辑.查找上一个选定项	Ctrl + Shift + F3	查找当前选定文本的上一匹配项，或插入点处的单词的上一匹配项。
编辑.转到查找组合框	CTRL + /	将插入点放置在“标准”工具栏上的“查找/命令”框中。
编辑.渐进式搜索	Ctrl + I	开始渐进式搜索。如果启动了渐进式搜索，但未键入任何字符，则会恢复上一模式。如果已找到文本，则搜索下一匹配项。
编辑.替换	Ctrl + H	在“查找和替换”对话框的“快速替换”选项卡中显示替换选项。
编辑.在文件中替换	Ctrl + Shift + H	在“查找和替换”对话框的“在文件中替换”选项卡中显示替换选项。
编辑.反向渐进式搜索	Ctrl + Shift + I	将渐进式搜索的搜索方向更改为，从文件尾开始向文件头搜索。
编辑.停止搜索	Alt + F3, S	停止当前的“在文件中查找”操作。
视图.查找符号结果	Ctrl + W, Ctrl + Q	显示“查找符号结果”窗口，该窗口显示符号搜索的匹配项。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 重构快捷键

下列键组合为用于执行[重构](#)操作的快捷键。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

命令名	快捷键	说明
重构.封装字段	Ctrl + R, C trl + E	显示“ 封装字段 ”对话框，利用它可以从现有字段创建属性，然后更新代码以引用新属性。
重构.提取接口	Ctrl + R, C trl + I	显示“ 提取接口 ”对话框，利用它可以使用从现有类、结构或接口派生的成员创建新接口。
重构.提取方法	Ctrl + R, C trl + M	显示“ 提取方法 ”对话框，利用它可以从现有方法的代码段创建新方法。
重构.将局部变量提升为参数	Ctrl + R, C trl + P	将变量从局部使用移动至方法、索引器或构造函数参数，并正确更新调用站点。有关更多信息，请参见 将局部变量提升为参数 。
重构.移除参数	Ctrl + R, C trl + V	显示“ 移除参数 ”对话框，该对话框可通过更改调用成员的任何位置处的声明，从方法、索引器或委托中移除参数。有关更多信息，请参见 移除参数 。
重构.重命名	F2 - 或 - Ctrl + R, C trl + R	显示“ 重命名 ”对话框，利用它可以重命名代码符号的标识符（如字段、局部变量、方法、命名空间、属性和类型）。
重构.重新排列参数	Ctrl + R, C trl + O	显示“ 重新排列参数 ”对话框，利用它可以为方法、索引器和委托更改参数的顺序。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 托管资源编辑器快捷键

只有在托管资源编辑器中进行编辑时才能使用下列快捷组合键。有关更多信息，请参见“[项目设计器”->“资源”页。](#)

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.编辑单元格	F2	在“其他”视图和“字符串”视图的选定单元格中切换到编辑模式。
编辑.移除	Delete	移除“文件”视图、“图像”视图、“图标”视图和“音频”视图中的选定文件。
编辑.移除行	Ctrl + Delete	删除“其他”视图和“字符串”视图中的选定行。
资源.音频	Ctrl + 4	将托管资源编辑器切换到“音频”视图，该视图会显示当前项目中的声音文件。显示文件的格式包括 .wav、.wma 和 .mp3。
资源.文件	Ctrl + 5	将托管资源编辑器切换到“文件”视图，该视图会显示在其他视图中找不到的文件。
资源.图标	Ctrl + 3	将托管资源编辑器切换到“图标”视图，该视图会显示当前项目中的图标 (*.ico) 文件。
资源.图像	Ctrl + 2	将托管资源编辑器切换到“图像”视图，该视图会显示当前项目中的图像文件。显示文件的格式包括 .bmp、.jpg 和 .gif。
资源.其他	Ctrl + 6	将托管资源编辑器切换到“其他”视图，其中显示一个设置网格，用于添加支持字符串序列化的其他类型。
资源.字符串	Ctrl + 1	将托管资源编辑器切换到“字符串”视图，其中以网格方式显示字符串，并包含字符串资源的“名称”、“值”和“注释”列。

请参见

概念

[Visual C# 2005 默认快捷键](#)

[.Resx 文件格式中的资源](#)

Visual C# 2005 方案 -> 代码段快捷键

下列快捷组合键可用于处理代码段。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.插入代码段	Ctrl + K, Ctrl + X	显示 代码段选择器 ，可以利用它通过 IntelliSense 选择代码段，然后在光标位置插入代码段。
编辑.外侧代码	Ctrl + K, Ctrl + S	显示 代码段选择器 ，可以利用它通过 IntelliSense 选择代码段，然后让该代码段环绕于选定文本周围。
工具.代码段管理器	Ctrl + K, Ctrl + B	显示 代码段管理器 ，可以利用它搜索代码段并将其插入文件中。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 类关系图快捷键

只有当您使用类关系图时，才能使用以下快捷键组合。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令名	快捷键	说明
类关系图.折叠	数字键区中的 -(减号)	折叠“类详细信息”窗口中展开的节点，或折叠关系图中选定的形状隔离舱。
类关系图.展开	数字键区中的 +(加号)	展开“类详细信息”窗口中折叠的节点，或展开关系图中选定的形状隔离舱。
编辑.删除	Ctrl + Delete	从类关系图中移除选定项。
编辑.展开折叠基类型列表	Shift + Alt + B	展开或折叠选定形状隔离舱中的基类型。 例如，如果“接口 1”继承自“接口 2”、“接口 3”和“接口 4”，则父接口将列在“接口 1”的形状隔离舱上。通过使用此命令，可以折叠继承的接口的列表，只显示有关由“接口 1”继承的基接口数的摘要信息。
编辑.定位到棒糖形	Shift + Alt + L	为形状隔离舱选择棒糖形接口。会在实现了一个或多个接口的形状中显示棒糖形。
编辑.从关系图中移除	Delete	从关系图中移除选定的形状隔离舱。
视图.查看代码	Enter - 或 - F7	对于选定项，打开相应文件并将插入点放置在正确位置。

请参见

概念

[Visual C# 2005 默认快捷键](#)

其他资源

[使用类关系图](#)

Visual C# 2005 方案 -> 书签窗口快捷键

下列快捷组合键仅当在[书签窗口](#)或编辑器中使用书签时才可用。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

命令名	快捷键	说明
编辑.清除书签	Ctrl + B, Ctrl + C	移除文档中的所有书签
编辑.启用书签	Ctrl + B, Ctrl + E	在当前文档中启用书签用法。
编辑.下一书签	Ctrl + B, Ctrl + N	移动到文档中下一个书签处。
编辑.上一书签	Ctrl + B, Ctrl + P	移动到上一书签。
编辑.切换书签	Ctrl + B, Ctrl + T	在文档中的当前行切换书签。
视图.书签窗口	Ctrl + W, Ctrl + B	显示“书签”窗口。

请参见

概念

[Visual C# 2005 默认快捷键](#)

Visual C# 2005 方案 -> 快捷键和字符串编辑器快捷键

在“快捷键”编辑器或“字符串”编辑器中可以使用下列快捷组合键。

注意

对话框中的可用选项以及显示的菜单命令的名称和位置可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上单击“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

命令	快捷键	说明
编辑.新建快捷键	Insert	为键盘快捷键添加新项。仅可用于“快捷键”编辑器。
编辑.新建字符串	Insert	在字符串表中添加新项。仅可用于“字符串”编辑器。
编辑.键入的下一个键	CTRL + W	显示“捕获下一个键”消息框，此消息框提示您按下要作为键盘快捷键使用的键。仅可用于“快捷键”编辑器。

请参见

概念

[Visual C# 2005 默认快捷键](#)

迁移到 Visual C#

本节向从其他编程语言进行迁移的开发人员介绍 C# 语法和概念。它还包含可用于将 Java 语言源代码转换为 C# 源代码的 Java Language Conversion Assistant 的参考文档。

本节内容

[C#\(针对 Java 开发人员\)](#)

将 C# 语言的语法和构造与 Java 语言进行比较。

[将 Java 应用程序转换为 Visual C#](#)

描述 Java Language Conversion Assistant, 它是一个用于将 Java 项目移植到 Visual C# 的工具。

[C#\(针对 C++ 开发人员\)](#)

比较 C# 语言与 C++ 语言的功能。

请参见

概念

[C# 编程指南](#)

其他资源

[Visual C#](#)

[Visual C# 入门](#)

C#(针对 Java 开发人员)

本节的主题提供对 C# 语言和 .NET Framework 的介绍。

本节内容

- [C# 编程语言\(针对 Java 开发人员\)](#)
- [C# 代码示例\(针对 Java 开发人员\)](#)
- [C# 应用程序类型\(针对 Java 开发人员\)](#)

请参见

概念

[Java Language Conversion Assistant](#)

其他资源

[Visual C# 入门](#)

C# 编程语言(针对 Java 开发人员)

本节讨论 C# 和 Java 编程语言之间的相似点和差异。

本节内容

[源文件约定\(C# 与 Java\)](#)

[数据类型\(C# 与 Java\)](#)

[运算符\(C# 与 Java\)](#)

[流控制\(C# 与 Java\)](#)

[循环语句\(C# 与 Java\)](#)

[类基础\(C# 与 Java\)](#)

[Main \(\) 方法和其他方法\(C# 与 Java\)](#)

[使用不确定数目的参数\(C# 与 Java\)](#)

[属性\(C# 与 Java\)](#)

[Struct\(C# 与 Java\)](#)

[数组](#)

[继承与派生类\(C# 与 Java\)](#)

[事件](#)

[运算符重载\(C# 与 Java\)](#)

[异常\(C# 与 Java\)](#)

[高级 C# 技术\(C# 与 Java\)](#)

[垃圾回收\(C# 与 Java\)](#)

[安全代码与不安全的代码\(C# 与 Java\)](#)

[摘要\(C# 与 Java\)](#)

相关章节

[C# 编程指南](#)

[Visual C#](#)

[迁移到 Visual C#](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

源文件约定(C# 与 Java)

包含 C# 类的文件的命名约定与 Java 文件的命名约定有所不同。在 Java 中，所有源文件的扩展名均为 `.java`。每个源文件均包含一个顶级公共类声明，并且该类的名称必须与文件名匹配。换句话说，使用公共范围声明的类 **Customer** 必须在名为 `Customer.java` 的源文件中定义。

C# 源代码用 `.cs` 扩展名表示。与 Java 不同，C# 源文件可以包含多个顶级公共类声明，并且文件名不必与任何类名称匹配。

顶级声明

在 Java 和 C# 中，源代码的开头为几个按一定顺序排列的顶级声明。Java 和 C# 程序中的声明仅有少许不同。

Java 中的顶级声明

在 Java 中，可以使用 **package** 关键字将类组合在一起。组合后的打包类必须在源文件第一行可执行代码中使用 **package** 关键字。如下所示，之后是访问其他包中的类所需的任何导入语句，然后是类声明：

```
package Acme;
import java.io.*;
class Customer
{
    ...
}
```

C# 中的顶级声明

C# 使用命名空间的概念，通过 **namespace** 关键字将相关类按逻辑方式组合在一起。这些组合类的作用与 Java 包的作用相似，并且同一个类可以出现在两个不同的命名空间内。若要访问在当前命名空间之外的某个命名空间中定义的类，请使用 **using** 指令，并在该指令后面加上命名空间名称，如下所示：

C#

```
using System.IO;

namespace Acme
{
    class Customer
    {
        // ...
    }
}
```

注意，可以在命名空间声明内放入 **using** 指令，在这种情况下，这些导入的命名空间将成为包含命名空间的一部分。

Java 不允许在同一个源文件中出现多个包。而 C# 允许在一个 `.cs` 文件中出现多个命名空间，如下所示：

C#

```
namespace AcmeAccounting
{
    public class GetDetails
    {
        // ...
    }
}

namespace AcmeFinance
{
    public class ShowDetails
    {
        // ...
    }
}
```

完全限定名称和命名空间别名

与 Java 一样, 不对命名空间使用 `using` 引用仍可以访问 .NET Framework 命名空间或用户定义的命名空间中的类, 方法是提供类的完全限定名称, 例如, 上例中的 `DataSet` 或 `AcmeAccounting.GetDetails`。

但完全限定名称可能因名称太长而难以处理, 在这种情况下, 可以使用 `using` 关键字指定一个简称或别名, 以提高代码的可读性。

在下面的代码中, 将创建一个别名, 以引用由某个虚构公司编写的代码:

C#

```
using DataTier = Acme.SQLCode.Client;

class OutputSales
{
    static void Main()
    {
        int sales = DataTier.GetSales("January");
        System.Console.WriteLine("January's Sales: {0}", sales);
    }
}
```

注意, 在 `WriteLine` 的语法中, 格式字符串中含有 `{x}`, 其中的 `x` 表示要插入到该位置的值在参数列表中的位置。假设 `GetSales` 方法返回 500, 则应用程序的输出应当为:

January's Sales: 500

预处理指令

与 C 和 C++ 类似, C# 包含预处理指令, 这些指令提供以下能力:有条件地跳过源文件中的节、报告错误和警告条件以及描绘源代码的不同区域。使用“预处理指令”一词仅仅是为了与 C 和 C++ 编程语言保持一致, 因为 C# 并不包括单独的预处理步骤。有关更多信息, 请参见 [C# 预处理器指令](#)。

请参见

概念

[C# 编程指南](#)

其他资源

[C# 编程语言\(针对 Java 开发人员\)](#)

数据类型(C# 与 Java)

本主题讨论 Java 和 C# 在数据的表示、分配和垃圾回收等方面的一些主要相同点和不同点。

复合数据类型

在 Java 和 C# 中，类作为有字段、方法和事件的复合数据类型这一概念是相似的。(有关类继承的概念在名为[继承与派生类\(C# 与 Java\)](#)的主题中单独讨论。)C# 引入结构的概念，结构是一种堆栈分配的复合数据类型，它不支持继承。在其他许多方面，结构与类非常相似。结构提供一种将相关字段和方法组合在一起的轻量方法，以便在紧凑循环和其他性能关键的方案中使用。

C# 使您能够创建一个在对类的实例进行垃圾回收前调用的析构函数方法。在 Java 中，可以使用 **finalize** 方法来包含代码，用于在将对象作为垃圾回收前清理资源。在 C# 中，此功能由类析构函数执行。析构函数就像是没有参数和前面不带颤化符(~)的构造函数。

内置数据类型

C# 提供 Java 中可用的所有数据类型，并增加了对无符号数字和新的 128 位高精度浮点类型的支持。

核心类库为 Java 中的每个基元数据类型提供了一个包装类，此包装类将基元数据类型表示为 Java 对象。例如，[Int32](#) 类包装 **int** 数据类型，[Double](#) 类包装 **double** 数据类型。

另一方面，C# 中的所有基元数据类型都是 [System](#) 命名空间中的对象。对于每个数据类型，提供了一个简称(或别名)。例如，**int** 是 [System.Int32](#) 的简称，而 **double** 是 [System.Double](#) 的简写。

下表提供了 C# 数据类型列表及其别名。如表所示，前八个数据类型对应于 Java 中可用的基元类型。但请注意，Java 的 **boolean** 在 C# 中称为 **bool**。

简称	.NET 类	类型	宽度	范围(位)
byte	Byte	无符号整数	8	0 到 255
sbyte	SByte	有符号整数	8	-128 到 127
int	Int32	有符号整数	32	-2,147,483,648 到 2,147,483,647
uint	UInt32	无符号整数	32	0 到 4294967295
short	Int16	有符号整数	16	-32,768 到 32,767
ushort	UInt16	无符号整数	16	0 到 65535
long	Int64	有符号整数	64	-922337203685477508 到 922337203685477507
ulong	UInt64	无符号整数	64	0 到 18446744073709551615
float	Single	单精度浮点型	32	-3.402823e38 至 3.402823e38
double	Double	双精度浮点型	64	-1.79769313486232e308 至 1.79769313486232e308
char	Char	单 Unicode 字符	16	文本中使用的 Unicode 符号
bool	Boolean	逻辑布尔值类型	8	True 或 False
object	Object	所有其他类型的基类型		

<code>string</code>	<code>String</code>	字符序列		
<code>decimal</code>	<code>Decimal</code>	精确小数类型或整型, 可以表示带有 29 个有效位的十进制数	128	$\pm 1.0 \times 10^{-28}$ 至 $\pm 7.9 \times 10^{28}$

因为 C# 将所有基元数据类型当作对象表示, 所以可以在基元数据类型上调用对象方法。例如:

C#

```
static void Main()
{
    int i = 10;
    object o = i;
    System.Console.WriteLine(o.ToString());
}
```

借助自动装箱和取消装箱完成此操作。有关更多信息, 请参见[装箱和取消装箱\(C# 编程指南\)](#)。

常数

Java 和 C# 均能够声明这样一个变量: 它的值在编译时指定, 在运行时不能更改。Java 使用 `final` 字段修饰符声明此类变量, 而 C# 则使用 `const` 关键字。除了 `const` 以外, C# 还提供 `readonly` 关键字以声明可以在运行时进行一次赋值(在声明语句中或在构造函数中)的变量。初始化以后, `readonly` 变量的值不能更改。当已单独编译的模块需要共享版本号等数据时, 可以使用 `readonly` 变量。如果模块 A 更新了, 并使用一个新的版本号进行了重新编译, 则模块 B 可以用这个新的常数值进行初始化, 而无需重新编译。

枚举

枚举用于对已命名常数进行分组, 与在 C 和 C++ 中的用法相似, 但不可用于 Java。下面的示例定义一个简单的 `Color` 枚举。

C#

```
public enum Color
{
    Green,      //defaults to 0
    Orange,     //defaults to 1
    Red,        //defaults to 2
    Blue        //defaults to 3
}
```

也可以将整数值分配给枚举, 如下面的枚举声明所示:

C#

```
public enum Color2
{
    Green = 10,
    Orange = 20,
    Red = 30,
    Blue = 40
}
```

下面的代码示例调用 `Enum` 类型的 `GetNames` 方法来显示枚举的可用常数。然后, 将值分配给枚举, 并显示此值。

C#

```
class TestEnums
{
    static void Main()
    {
        System.Console.WriteLine("Possible color choices: ");

        //Enum.GetNames returns a string array of named constants for the enum.
        foreach(string s in System.Enum.GetNames(typeof(Color)))
```

```

    {
        System.Console.WriteLine(s);
    }

    Color favorite = Color.Blue;

    System.Console.WriteLine("Favorite Color is {0}", favorite);
    System.Console.WriteLine("Favorite Color value is {0}", (int) favorite);
}
}

```

输出

```

Possible color choices:
Green
Orange
Red
Blue
Favorite Color is Blue
Favorite Color value is 3

```

字符串

Java 和 C# 中的字符串类型的行为相似，只有细微的差异。两种字符串类型都是不可变的，意味着一旦创建了字符串，字符串的值就无法更改。两个实例中的方法看上去修改了字符串的实际内容，实际上创建并返回了一个新字符串，而原始字符串保持不变。C# 和 Java 中比较字符串值的过程有所不同。若要在 Java 中比较字符串值，则开发人员需要在字符串类型上调用 **equals** 方法，原因是默认情况下 == 运算符会比较引用类型。在 C# 中，开发人员可以直接使用 == 或 != 运算符来比较字符串值。尽管在 C# 中字符串是引用类型，但在默认情况下 == 和 != 运算符将比较字符串值而不是引用。

和在 Java 中一样，C# 开发人员不应使用字符串类型来串连字符串，以避免在每次串连字符串时创建新字符串类所产生的开销。相反，开发人员可以使用 [StringBuilder](#) 类，它与 Java 的 [StringBuffer](#) 类在功能上等效。

字符串文本

C# 能够避免在字符串常数中使用转义序列，如用于制表符的 "\t" 或用于反斜杠字符的 "\"。若要达到此目的，只需在分配字符串值之前使用 @ 符号声明原义字符串。下面的示例演示如何使用转义字符以及如何分配字符串文本：

C#

```

static void Main()
{
    //Using escaped characters:
    string path1 = @"\FileShare\Directory\file.txt";
    System.Console.WriteLine(path1);

    //Using String Literals:
    string path2 = @"\\FileShare\Directory\file.txt";
    System.Console.WriteLine(path2);
}

```

转换和强制转换

Java 和 C# 对数据类型的自动转换和强制转换遵循相似的规则。

和 Java 一样，C# 支持隐式和显式类型转换。如果是扩大转换，则为隐式转换。例如，下面从 **int** 到 **long** 的转换为隐式转换，与在 Java 中相同：

C#

```

int int1 = 5;
long long1 = int1; //implicit conversion

```

下面是 .NET Framework 数据类型之间的隐式转换列表：

源类型	目标类型
Byte	short、ushort、int、uint、long、ulong、float、double 或 decimal
Sbyte	short、int、long、float、double 或 decimal
Int	long、float、double 或 decimal
Uint	long、ulong、float、double 或 decimal
Short	int、long、float、double 或 decimal
Ushort	int、uint、long、ulong、float、double 或 decimal
Long	float、double 或 decimal
Ulong	float、double 或 decimal
Float	double
Char	ushort、int、uint、long、ulong、float、double 或 decimal

使用与 Java 相同的语法强制转换要显式转换的表达式：

C#

```
long long2 = 5483;
int int2 = (int)long2; //explicit conversion
```

下表列出了显式转换。

源类型	目标类型
Byte	sbyte 或 char
Sbyte	byte、ushort、uint、ulong 或 char
Int	sbyte、byte、short、ushort、uint、ulong 或 char
Uint	sbyte、byte、short、ushort、int 或 char
Short	sbyte、byte、ushort、uint、ulong 或 char
Ushort	sbyte、byte、short 或 char
Long	sbyte、byte、short、ushort、int、uint、ulong 或 char
Ulong	sbyte、byte、short、ushort、int、uint、long 或 char
Float	sbyte、byte、short、ushort、int、uint、long、ulong、char 或 decimal
Double	sbyte、byte、short、ushort、int、uint、long、ulong、char、float 或 decimal
Char	sbyte、byte 或 short

Decimal	sbyte、byte、short、ushort、int、uint、long、ulong、char、float 或 double
---------	---

值与引用类型

C# 支持两种变量类型：

- 值类型

这些是内置基元数据类型(如 char、int 和 float)以及用 struct 声明的用户定义类型。

- 引用类型

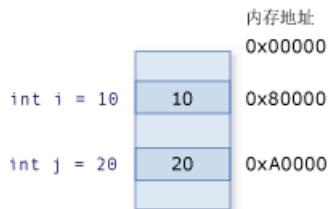
从基元类型构造的类和其他复杂的数据类型。这种类型的变量不包含类型的实例而仅包含对实例的引用。

如果创建两个值类型变量 i 和 j(如下所示)，则 i 和 j 完全相互独立：

C#

```
int i = 10;
int j = 20;
```

为它们指定了独立的内存位置：



如果更改这两个变量中其中一个的值，另一个变量当然不受影响。例如，如果具有如下形式的表达式，则这两个变量之间仍然没有关联：

C#

```
int k = i;
```

也就是说，如果更改 i 的值，则 k 将保留赋值时 i 具有的值。

C#

```
i = 30;

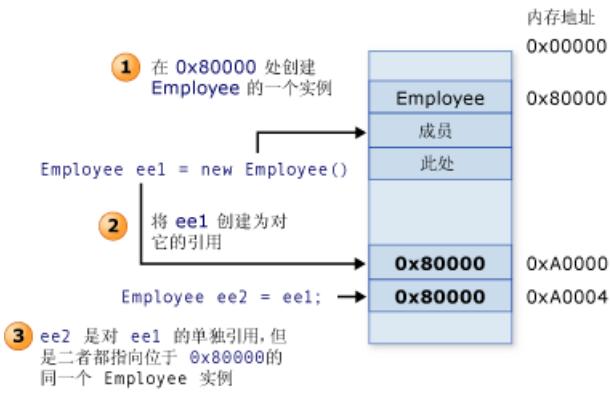
System.Console.WriteLine(i.ToString()); // 30
System.Console.WriteLine(k.ToString()); // 10
```

但是，引用类型的行为则不同。例如，可以声明如下所示的两个变量：

C#

```
Employee ee1 = new Employee();
Employee ee2 = ee1;
```

现在，因为类在 C# 中为引用类型，所以 ee1 被视为对 Employee 的引用。在前面的两行中，第一行在内存中创建 Employee 的一个实例，并设置 ee1 以引用该实例。因此，将 ee2 设置为等于 ee1 时，它包含了对内存中的类的重复引用。如果现在更改 ee2 上的属性，则 ee1 上的属性将反映这些更改，因为两者都指向内存中的相同对象，如下所示：



装箱和取消装箱

将值类型转换为引用类型的过程称为装箱。反之，将引用类型转换为值类型的过程则称为取消装箱。下面的代码说明了这一点：

C#

```
int i = 123;           // a value type
object o = i;          // boxing
int j = (int)o;        // unboxing
```

Java 要求手动执行这种转换。可以通过构造这种对象或装箱，将基元数据类型转换为包装类的对象。同样地，可以通过在这种对象上调用合适的方法或取消装箱，从包装类对象中提取基元数据类型的值。有关装箱和取消装箱的更多信息，请参见[装箱转换\(C# 编程指南\)](#)和[取消装箱转换\(C# 编程指南\)](#)。

[请参见](#)

[参考](#)

[数据类型\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C#](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

运算符(C# 与 Java)

C# 提供 Java 支持的所有适用的运算符，如下表所示。在表的最后，您会看到在 C# 中可用但 Java 中没有的一些新运算符：

类别	符号
一元	<code>++ -- + - ! ~ ()</code>
乘法	<code>* / %</code>
加法	<code>+ -</code>
移位	<code><< >></code>
关系	<code>< > <= >= instanceof</code>
相等	<code>== !=</code>
逻辑“与”	<code>&</code>
逻辑 XOR	<code>^</code>
逻辑“或”	<code> </code>
条件 AND	<code>&&</code>
条件 OR	<code> </code>
条件运算	<code>? :</code>
赋值	<code>= *= /= %= += -= <<= >>= &= ^= =</code>
操作数类型	<code>typeof</code>
操作数大小	<code>sizeof</code>
执行溢出检查	<code>checked</code>
取消溢出检查	<code>unchecked</code>

唯一不能在 C# 中使用的 Java 运算符是移位运算符 (`>>>`)。因为 Java 中缺少无符号的变量，所以需要这一运算符处理需要向右移位以在最高有效比特位中插入 1 这种情况。

C# 支持无符号变量，因此，C# 只需要标准的 `>>` 运算符。此运算符根据操作数是否有符号会产生不同的结果。将一个无符号数向右移位会在最高有效比特位中插入 0，而将一个有符号数向右移位会复制前面的最高有效比特位。

Checked 和 Unchecked 运算符

如果结果大小比分配给所用数据类型的位数大很多，算术运算将导致溢出。对给定的整数算术运算使用 `checked` 和 `unchecked` 关键字可以检查或忽略此类溢出。如果表达式是一个使用 `checked` 的常数表达式，就会在编译时产生错误。

下面这个简单的示例演示了这些运算符：

C#

```
class TestCheckedAndUnchecked
{
    static void Main()
```

```
{  
    short a = 10000;  
    short b = 10000;  
  
    short c = (short)(a * b);           // unchecked by default  
    short d = unchecked((short)(10000 * 10000)); // unchecked  
    short e = checked((short)(a * b));      // checked - run-time error  
  
    System.Console.WriteLine(10000 * 10000); // 100000000  
    System.Console.WriteLine(c);          // -7936  
    System.Console.WriteLine(d);          // -7936  
    System.Console.WriteLine(e);          // no result  
}  
}
```

在这段代码中, `unchecked` 运算符绕过编译时错误, 否则下列语句可能会导致编译时错误:

C#

```
short d = unchecked((short)(10000 * 10000)); // unchecked
```

下面的表达式在默认情况下 `unchecked`, 因此值溢出后无提示:

C#

```
short c = (short)(a * b);           // unchecked by default
```

我们可以在运行时使用 `checked` 运算符强制检查表达式是否发生溢出:

C#

```
short e = checked((short)(a * b));      // checked - run-time error
```

将前两个值赋给 `d` 和 `c`, 会在程序运行时无提示地溢出值 -7936, 但是在尝试相乘以获取使用 `checked()` 的 `e` 的值时, 此程序将引发 `OverflowException`。

注意

也可以通过使用命令行编译器开关 (`/checked`) 控制是否检查代码块的算术溢出, 或直接在 Visual Studio 中基于每个项目进行控制。

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 编程语言\(针对 Java 开发人员\)](#)

流控制(C# 与 Java)

Java 和 C# 中的流控制语句(如 **if else** 和 **switch** 语句)非常相似。

分支语句

分支语句根据特定条件更改运行时的程序执行流。

if, else 和 else if

在 Java 和 C# 这两种语言中, 这些语句完全相同。

switch 语句

在两种语言中, **switch** 语句均提供条件性多重分支操作。但是, 它们之间也有不同。在 Java 中, 您可以“跳过”一个 case 而执行下一个 case, 除非此 case 的末尾使用了 **break** 语句。而 C# 要求在每个 case 的末尾都使用 **break** 或 **goto** 语句。否则, 编译器将产生以下错误:

控制不能从一个 case 标签贯穿到另一个 case 标签。

注意:如果 case 没有指定任何要在匹配此 case 时执行的代码, 控制将贯穿到随后的 case。如果 **switch** 语句中使用了 **goto**, 那么您只能跳到同一个 switch 中的另一个 case 块。如果要跳到默认 case, 应使用 **goto default**;否则, 应使用 **goto case cond**, 此处的 **cond** 就是您希望跳到的那个 case 的匹配条件。与 Java **switch** 的另一点区别是:在 Java 中, **switch** 语句只能使用整型变量, 而在 C# 中可以使用字符串变量。

例如, 以下代码在 C# 中为有效代码, 而在 Java 中为无效代码:

C#

```
static void Main(string[] args)
{
    switch (args[0])
    {
        case "copy":
            //...
            break;

        case "move":
            //...
            goto case "delete";

        case "del":
        case "remove":
        case "delete":
            //...
            break;

        default:
            //...
            break;
    }
}
```

goto 的返回

在 Java 中, **goto** 是不能实现的保留关键字。但是, 您可以使用具有 **break** 或 **continue** 的标签语句来实现与 **goto** 类似的用途。

C# 允许 **goto** 语句跳到标签语句处。但是请注意, 要跳到某个特定的标签处, **goto** 语句必须在此标签的范围内。换言之, 尽管 **goto** 可以从一个类中跳出, 从而跳到此类之外或退出 **try...catch** 语句中的 **finally** 块, 但不能使用它跳到某个语句块内。大多数情况下, 建议不要使用 **goto**, 因为它不是一种面向对象编程的好做法。

请参见

概念

[C# 编程指南](#)

其他资源

Visual C#

C# 编程语言(针对 Java 开发人员)

循环语句(C# 与 Java)

循环语句重复指定的代码块，直到满足指定的条件。

for 循环

在 C# 和 Java 中，**for** 循环的语法和操作相同：

C#

```
for (int i = 0; i<=9; i++)
{
    System.Console.WriteLine(i);
}
```

foreach 循环

C# 引入了一种称为 **foreach** 循环的新循环类型，它与 Visual Basic 的 **For Each** 相似。对于支持 **IEnumerable** 接口的容器类（如数组），可以使用 **foreach** 循环来循环访问其中的每一项。下面的代码演示如何使用 **foreach** 语句输出数组的内容：

C#

```
static void Main()
{
    string[] arr= new string[] {"Jan", "Feb", "Mar"};
    foreach (string s in arr)
    {
        System.Console.WriteLine(s);
    }
}
```

有关更多信息，请参见[数组\(C# 与 Java\)](#)。

while 和 do...while 循环

在这两种语言中，**while** 和 **do...while** 语句的语法和操作是相同的：

C#

```
while (condition)
{
    // statements
}
```

C#

```
do
{
    // statements
}
while(condition); // Don't forget the trailing ; in do...while loops
```

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

类基础(C# 与 Java)

以下各节对 C# 和 Java 的修饰符进行了比较。

访问修饰符

C# 修饰符与 Java 修饰符十分相似，只有几个小小的差异。可以使用访问修饰符声明类的每个成员或类本身，从而定义允许访问的范围。不是在其他类内部声明的类只能指定公共修饰符或内部修饰符。嵌套的类与其他类成员一样，可以指定下面五个访问修饰符中的任何一个：

- **public**

对所有类都可见。

- **protected**

只对派生类可见。

- **private**

只在给定的类中可见。

- **internal**

只在相同的程序集内可见。

- **protected internal**

只对当前程序集或从包含类派生的类型可见。

Public、Protected 和 Private 修饰符

public 修饰符表示成员可在任何地方(类的内部和外部)使用。**protected** 修饰符指示只能从包含类或从包含类派生的类内部进行访问。**private** 修饰符表示只能从包含类型的内部进行访问。在 C# 中，默认的访问修饰符为 Private，而在 Java 中，默认情况下，可在包含包内的任何位置进行访问。

Internal 修饰符

只能在当前程序集内部访问 **internal** 项。.NET Framework 中的程序集差不多相当于 Java 的 JAR 文件，它表示可从其中构造其他程序的构造块。

Protected Internal 修饰符

protected internal 项只对当前程序集或从包含类派生的类型可见。

Sealed 修饰符

类声明中带有 **sealed** 修饰符的类与抽象类相反：它不能被继承。您可以将一个类标记为 Sealed，以防止其他类重写其功能。自然，密封类不可能是抽象类。还要注意，**struct** 是隐式密封的；因此，它不能被继承。**sealed** 修饰符相当于在 Java 中用 **final** 关键字标记一个类。

Readonly 修饰符

若要在 C# 中定义常数，请使用 **const** 或 **readonly** 修饰符来代替 Java 的 **final** 关键字。C# 中这两个修饰符的主要不同在于，**const** 项在编译时处理，而 **readonly** 字段的值则是在运行时指定。这意味着可以在类构造函数以及声明中对 **readonly** 字段赋值。例如，下面的类声明了一个名为 IntegerVariable 的 **readonly** 变量，它在类构造函数中初始化：

C#

```
public class SampleClass
{
    private readonly int intConstant;

    public SampleClass () //constructor
    {
        // You are allowed to set the value of the readonly variable
        // inside the constructor
        intConstant = 5;
    }
}
```

```
public int IntegerConstant
{
    set
    {
        // You are not allowed to set the value of the readonly variable
        // anywhere else but inside the constructor

        // intConstant = value; // compile-time error
    }
    get
    {
        return intConstant;
    }
}
class TestSampleClass
{
    static void Main()
    {
        SampleClass obj= new SampleClass();

        // You cannot perform this operation on a readonly field.
        obj.IntegerConstant = 100;

        System.Console.WriteLine("intConstant is {0}", obj.IntegerConstant); // 5
    }
}
```

如果 **readonly** 修饰符应用于静态字段，它应该在类的静态构造函数中被初始化。

[请参见](#)

[参考](#)

[访问修饰符 \(C# 编程指南\)](#)

[常量 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 编程语言 \(针对 Java 开发人员\)](#)

Main () 方法和其他方法(C# 与 Java)

本节讨论方法和如何通过引用和通过值来传递方法参数。

Main () 方法

每个 C# 应用程序都必须包含一个 **Main** 方法，以指定从哪里开始执行程序。在 C# 中，**Main** 的首字母大写，而 Java 使用小写的 **main**。

Main 只能返回 **int** 或 **void**，并有一个代表命令行参数的可选字符串数组参数：

C#

```
static int Main(string[] args)
{
    //...
    return 0;
}
```

对于包含任何传入的命令行参数的字符串数组参数，其工作方式与在 Java 中的工作方式相同。因此，`args[0]` 指定第一个命令行参数，`args[1]` 表示第二个参数，依此类推。与 C++ 不同，`args` 数组不包含 EXE 文件名称。

其他方法

向某个方法传递参数时，可以通过值或通过引用传递参数。值参数简单地采用任意变量的值，在方法中使用。因此，调用代码中的变量值不受对方法内的参数所执行的操作的影响。

而引用参数指向在调用代码中声明的变量，因此，通过引用传递时方法将修改此变量的内容。

通过引用传递

在 Java 和 C# 中，引用某个对象的方法参数总是通过引用传递，而基元数据类型参数则通过值传递。

在 C# 中，所有参数在默认情况下均通过值传递。若要通过引用传递，您需要指定关键字 **ref** 或 **out**。这两个关键字的区别在于参数初始化的不同。**ref** 参数必须在使用前进行初始化，而 **out** 参数不必在传递前进行显式初始化，并且它将忽略以前的任何值。

ref 关键字

如果想让被调用方法永久更改用作参数的变量的值时，请在参数中指定 **ref** 关键字。这种做法传递的不是调用中所使用的变量的值，而是对此变量本身的引用。方法随后会处理此引用，因此在方法执行期间对此参数所做的更改将保存到用作此方法的参数的原始变量中。

下面的代码在 `Add` 方法中演示了此功能，其中的第二个 `int` 参数就是使用 **ref** 关键字通过引用传递的：

C#

```
class TestRef
{
    private static void Add(int i, ref int result)
    {
        result += i;
        return;
    }

    static void Main()
    {
        int total = 20;
        System.Console.WriteLine("Original value of 'total': {0}", total);

        Add(10, ref total);
        System.Console.WriteLine("Value after calling Add(): {0}", total);
    }
}
```

此示例很简单，其输出表明对结果参数所做的更改已在 Add 方法调用所使用的 total 变量中体现出来：

```
Original value of 'total': 20
Value after calling Add(): 30
```

这是因为结果参数引用了调用代码中 total 变量所占用的实际内存位置。类的属性不是变量，因此不能直接用作 ref 参数。

调用方法时，ref 关键字必须位于参数前面，并且必须位于方法声明之中。

out 关键字

out 关键字的作用与 ref 关键字非常相似，并且对使用 out 关键字声明的参数所做的更改将在方法之外体现出来。out 关键字与 ref 关键字有两点不同：(1) 方法中将忽略 out 参数的任何初始值；(2) 必须在方法调用过程中为 out 参数赋值：

C#

```
class TestOut
{
    private static void Add(int i, int j, out int result)
    {
        // The following line would cause a compile error:
        // System.Console.WriteLine("Initial value inside method: {0}", result);

        result = i + j;
        return;
    }

    static void Main()
    {
        int total = 20;
        System.Console.WriteLine("Original value of 'total': {0}", total);

        Add(33, 77, out total);
        System.Console.WriteLine("Value after calling Add(): {0}", total);
    }
}
```

在此示例中，Add 方法的第三个参数就是使用 out 关键字进行声明的，并且在调用此方法时还需要对该参数使用 out 关键字。输出将为：

```
Original value of 'total': 20
Value after calling Add(): 110
```

总而言之，如果想让方法修改现有变量，请使用 ref 关键字；若要返回在方法内生成的值，请使用 out 关键字。当方法为调用代码生成多个结果值时，通常将 out 关键字与方法的返回值结合起来使用。

请参见

参考

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

[传递参数 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 编程语言 \(针对 Java 开发人员\)](#)

使用不确定数目的参数(C# 与 Java)

C# 允许您在声明某个方法时通过指定 `params` 关键字来向此方法发送数目不确定的参数。变量列表也可包含常规参数，但请注意：使用 `params` 关键字声明的参数必须放在最后。它采取长度可变的数组形式，并且每个方法只能有一个 `params` 参数。

当编译器试图解析某个方法调用时，它会查找参数列表与所调用的方法相匹配的那个方法。如果无法找到与参数 (Argument) 列表相匹配的方法重载，但有一个具有相应类型的 `params` 参数 (Parameter) 的匹配版本，则将调用此方法，将其他参数 (Argument) 放入一个数组中。

下面的示例说明了这一点：

C#

```
class TestParams
{
    private static void Average(string title, params int[] values)
    {
        int sum = 0;
        System.Console.Write("Average of {0} (", title);

        for (int i = 0; i < values.Length; i++)
        {
            sum += values[i];
            System.Console.Write(values[i] + ", ");
        }
        System.Console.WriteLine("): {0}", (float)sum/values.Length);
    }
    static void Main()
    {
        Average ("List One", 5, 10, 15);
        Average ("List Two", 5, 10, 15, 20, 25, 30);
    }
}
```

以上示例中，方法 `Average` 是使用类型为整型数组的 `params` 参数声明的，这样您便可以用任意数目的变量来调用此方法。输出结果如下：

```
Average of List One (5, 10, 15, ): 10
Average of List Two (5, 10, 15, 20, 25, 30, ): 17.5
```

如果要允许使用不确定的不同类型的参数，可以指定 `Object` 类型的 `params` 参数。

请参见

参考

[将数组作为参数传递\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 编程语言\(针对 Java 开发人员\)](#)

属性(C# 与 Java)

在 C# 中，属性是类、结构或接口的命名成员，它提供了一种通过 **get** 和 **set** 访问器方法访问私有字段的巧妙方法。

下面的代码示例为 `Animal` 类声明 `Species` 属性，该属性将对私有变量 `name` 的访问抽象化：

C#

```
public class Animal
{
    private string name;

    public string Species
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

通常，属性的名称与它所访问的内部成员的名称相同，但是第一个字母要大写，如以上示例中的 `Name`，否则内部成员将带一个 `_` 前缀。此外，请注意 **set** 访问器中所使用的隐式参数 `value`；该参数具有基础成员变量的类型。

实际上，访问器在内部表示为 `get_X()` 和 `set_X()` 方法，以保持与基于 .NET Framework 的语言（它们不支持访问器）的兼容性。定义属性后，可以非常容易地获取或设置其值：

C#

```
class TestAnimal
{
    static void Main()
    {
        Animal animal = new Animal();
        animal.Species = "Lion";           // set accessor
        System.Console.WriteLine(animal.Species); // get accessor
    }
}
```

如果一个属性仅有 **get** 访问器，则它为只读属性。如果它仅有 **set** 访问器，则为只写属性。如果它同时具有这两种访问器，则为读写属性。

[请参见](#)

[参考](#)

[属性\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

Struct(C# 与 Java)

C# 支持 **struct** 关键字，它也是一个起源于 C 但在 Java 中不可用的项。可以将 **struct** 想像为一个轻量类。虽然 **structs** 可包含构造函数、常量、字段、方法、属性、索引器、运算符和嵌套类型，但是多数情况下，它们仅用于封装由相关字段组成的组。因为结构是值类型，所以它们的分配效率要比类略高些。**structs** 与类的不同之处在于，它们既不能是抽象的，也不支持实现继承。

在下面的示例中，用 **new** 关键字初始化 **struct**（调用默认的无参数构造函数），然后设置实例的成员。

C#

```
public struct Customer
{
    public int ID;
    public string Name;

    public Customer(int customerID, string customerName)
    {
        ID = customerID;
        Name = customerName;
    }
}

class TestCustomer
{
    static void Main()
    {
        Customer c1 = new Customer(); //using the default constructor

        System.Console.WriteLine("Struct values before initialization:");
        System.Console.WriteLine("ID = {0}, Name = {1}", c1.ID, c1.Name);
        System.Console.WriteLine();

        c1.ID = 100;
        c1.Name = "Robert";

        System.Console.WriteLine("Struct values after initialization:");
        System.Console.WriteLine("ID = {0}, Name = {1}", c1.ID, c1.Name);
    }
}
```

输出

编译和运行上述代码时，其输出结果显示 **struct** 变量已按默认值初始化。**int** 变量初始化为 0，**string** 变量初始化为空字符串：

Struct values before initialization:

ID = 0, Name =

Struct values after initialization:

ID = 100, Name = Robert

请参见

[任务](#)

[“结构”示例](#)

[概念](#)

[C# 编程指南](#)

[结构\(C# 编程指南\)](#)

[其他资源](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

数组 (C# 与 Java)

数组是具有相同数据类型的项的有序集合。要访问数组中的某个项，需要同时使用数组名称及该项与数组起点之间的偏移量。在 C# 中，声明和使用数组的方法与 Java 有一些重要区别。

一维数组

一维数组以线性方式存储固定数目的项，只需一个索引值即可标识任意一个项。在 C# 中，数组声明中的方括号必须跟在数据类型后面，且不能放在变量名称之后，而这在 Java 中是允许的。因此，类型为 integers 的数组应使用以下语法声明：

C#

```
int[] arr1;
```

下面的声明在 C# 中无效：

C#

```
//int arr2[]; //compile error
```

声明数组后，可以使用 **new** 关键字设置其大小，这一点与 Java 相同。下面的代码声明数组引用：

C#

```
int[] arr;
arr = new int[5]; // create a 5 element integer array
```

然后，可以使用与 Java 相同的语法访问一维数组中的元素。C# 数组索引也是从零开始的。下面的代码访问上面数组中的最后一个元素：

C#

```
System.Console.WriteLine(arr[4]); // access the 5th element
```

初始化

C# 数组元素可以在创建时使用与 Java 相同的语法进行初始化：

C#

```
int[] arr2Lines;
arr2Lines = new int[5] {1, 2, 3, 4, 5};
```

但 C# 初始值设定项的数目必须与数组大小完全匹配，这与 Java 不同。可以使用此功能在同一行中声明并初始化 C# 数组：

C#

```
int[] arr1Line = {1, 2, 3, 4, 5};
```

此语句创建一个数组，其大小等于初始值设定项的数目。

在程序循环中初始化

在 C# 中初始化数组的另一个方法是使用 **for** 循环。下面的循环将数组的每个元素都设置为零：

C#

```
int[] TaxRates = new int[5];
```

```
for (int i=0; i<TaxRates.Length; i++)
{
    TaxRates[i] = 0;
}
```

交错数组

C# 和 Java 都支持创建交错(非矩形)数组, 即每一行包含的列数不同的数组。例如, 在下面的交错数组中, 第一行有四项, 而第二行有三项:

C#

```
int[][] jaggedArray = new int[2][];
jaggedArray[0] = new int[4];
jaggedArray[1] = new int[3];
```

多维数组

可以使用 C# 创建规则的多维数组, 多维数组类似于同类型值的矩阵。虽然 Java 和 C# 都支持交错数组, 但 C# 还支持多维数组(数组的数组)。

使用以下语法声明多维矩形数组:

C#

```
int[,] arr2D; // declare the array reference
float[,,,] arr4D; // declare the array reference
```

声明之后, 可以按如下方式为数组分配内存:

C#

```
arr2D = new int[5,4]; // allocate space for 5 x 4 integers
```

然后, 可以使用以下语法访问数组的元素:

C#

```
arr2D[4,3] = 906;
```

由于数组是从零开始的, 因此此行将第四行第五列中的元素设置为 906。

初始化

可以使用以下一种方法, 在同一个语句中创建、设置并初始化多维数组:

C#

```
int[,] arr4 = new int [2,3] { {1,2,3}, {4,5,6} };
int[,] arr5 = new int [,] { {1,2,3}, {4,5,6} };
int[,] arr6 = { {1,2,3}, {4,5,6} };
```

在程序循环中初始化

可以使用此处所示的嵌套循环初始化数组中的所有元素:

C#

```
int[,] arr7 = new int[5,4];
for(int i=0; i<5; i++)
{
```

```

    for(int j=0; i<4; j++)
    {
        arr7[i,j] = 0; // initialize each element to zero
    }
}

```

System.Array 类

在 .NET Framework 中，数组是作为 [Array](#) 类的实例实现的。此类提供了许多有用的方法，如 [Sort](#) 和 [Reverse](#)。

下面的示例演示了使用这些方法是多么的简单。首先，使用 [Reverse](#) 方法将数组元素反转，然后使用 [Sort](#) 方法对它们进行排序：

C#

```

class ArrayMethods
{
    static void Main()
    {
        // Create a string array of size 5:
        string[] employeeNames = new string[5];

        // Read 5 employee names from user:
        System.Console.WriteLine("Enter five employee names:");
        for(int i=0; i<employeeNames.Length; i++)
        {
            employeeNames[i]= System.Console.ReadLine();
        }

        // Print the array in original order:
        System.Console.WriteLine("\nArray in Original Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0} ", employeeName);
        }

        // Reverse the array:
        System.Array.Reverse(employeeNames);

        // Print the array in reverse order:
        System.Console.WriteLine("\n\nArray in Reverse Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0} ", employeeName);
        }

        // Sort the array:
        System.Array.Sort(employeeNames);

        // Print the array in sorted order:
        System.Console.WriteLine("\n\nArray in Sorted Order:");
        foreach(string employeeName in employeeNames)
        {
            System.Console.Write("{0} ", employeeName);
        }
    }
}

```

输出

Enter five employee names:

Luca

Angie

Brian

Kent

Beatrix

Array in Original Order:

Luca Angie Brian Kent Beatrix

Array in Reverse Order:

Beatrix Kent Brian Angie Luca

Array in Sorted Order:

Angie Beatrix Brian Kent Luca

请参见

概念

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

其他资源

[C# 编程语言 \(针对 Java 开发人员\)](#)

继承与派生类(C# 与 Java)

您可以通过创建一个从现有类派生的新类来扩展现有类的功能。此派生类继承基类的属性，而且您可以根据需要添加或重写方法及属性。

在 C# 中，继承及接口实现均由：运算符定义，此运算符与 Java 中的 **extends** 和 **implements** 等效。在类声明中，基类应始终位于最左侧。

与 Java 一样，C# 也不支持多重继承，这意味着这些类不能从多个类中继承。不过，您可以使用接口实现此目的，具体操作方法与 Java 中的操作方法相同。

下面的代码使用两个私有成员变量 `x` 和 `y` 定义一个名为 `CoOrds` 的类，`x` 和 `y` 表示该点的位置。这两个变量分别通过名为 `X` 和 `Y` 属性访问：

C#

```
public class CoOrds
{
    private int x, y;

    public CoOrds() // constructor
    {
        x = 0;
        y = 0;
    }

    public int X
    {
        get { return x; }
        set { x = value; }
    }

    public int Y
    {
        get { return y; }
        set { y = value; }
    }
}
```

按如下方式从 `CoOrds` 类派生一个名为 `ColorCoOrds` 的新类：

C#

```
public class ColorCoOrds : CoOrds
```

这样，`ColorCoOrds` 继承了基类的所有字段和方法，您可以根据需要向基类添加新的字段和方法，以在此派生类中提供其他功能。在此示例中，您将添加一个私有成员和多个访问器，以便向该类中添加颜色：

C#

```
public class ColorCoOrds : CoOrds
{
    private System.Drawing.Color screenColor;

    public ColorCoOrds() // constructor
    {
        screenColor = System.Drawing.Color.Red;
    }

    public System.Drawing.Color ScreenColor
    {
        get { return screenColor; }
        set { screenColor = value; }
    }
}
```

```
}
```

派生类的构造函数隐式调用基类的构造函数(在 Java 术语中, 称为超类)。在继承中, 所有基类构造函数先于派生类构造函数获得调用, 并按照这些类在类层次结构中出现的先后顺序调用。

在类型上强制转换为基类

与在 Java 中一样, 不能使用对基类的引用访问派生类的成员和方法, 即使此基类引用可能包含对此派生类型的某个对象的有效引用。

您可以使用对此派生类型的引用来隐式引用某个派生类:

C#

```
ColorCoOrds color1 = new ColorCoOrds();
CoOrds coords1 = color1;
```

在此代码中, 基类引用 `coords1` 包含了 `color1` 引用的副本。

基关键字

您可以在子类中使用 `base` 关键字访问基类成员, 即使当这些基成员在超类中被重写时也可执行此操作。例如, 您可以创建一个派生类, 该派生类中包含一个签名与基类中的签名相同的方法。如果该方法以 `new` 关键字开头, 则表示此方法是属于该派生类的全新方法。您还可以使用基关键字再提供一个方法, 用于访问基类中的原始方法。

例如, 假设 `CoOrds` 基类有一个名为 `Invert()` 的方法, 它用于交换 `x` 和 `y` 坐标。您可以使用下列代码在 `ColorCoOrds` 派生类中提供此方法的替换方法:

C#

```
public new void Invert()
{
    int temp = X;
    X = Y;
    Y = temp;
    screenColor = System.Drawing.Color.Gray;
}
```

正如您所看到的, 此方法先交换 `x` 和 `y`, 然后将坐标的颜色设置为灰色。您可以为此方法提供对基实现的访问, 方法是在 `ColorCoOrds` 中再创建一个方法(如以下方法):

C#

```
public void BaseInvert()
{
    base.Invert();
}
```

接着通过调用 `BaseInvert()` 方法在 `ColorCoOrds` 对象中调用基方法。

C#

```
ColorCoOrds color1 = new ColorCoOrds();
color1.BaseInvert();
```

请记住, 如果先将对基类的引用分配给 `ColorCoOrds` 的一个实例, 然后再访问其方法, 也可以得到同样的效果:

C#

```
CoOrds coords1 = color1;
coords1.Invert();
```

选择构造函数

基类对象总是在任何派生类之前构建，因此，基类构造函数先于派生类构造函数获得执行。如果基类有多个构造函数，则派生类可以决定要调用哪个构造函数。例如，若要添加第二个构造函数，您可以按如下方式修改 `CoOrds` 类：

C#

```
public class CoOrds
{
    private int x, y;

    public CoOrds()
    {
        x = 0;
        y = 0;
    }

    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}
```

接着可以使用基关键字更改 `ColorCoOrds` 类以使用某个特定的可用构造函数：

C#

```
public class ColorCoOrds : CoOrds
{
    public System.Drawing.Color color;

    public ColorCoOrds() : base ()
    {
        color = System.Drawing.Color.Red;
    }

    public ColorCoOrds(int x, int y) : base (x, y)
    {
        color = System.Drawing.Color.Red;
    }
}
```

在 Java 中，此功能是通过 `super` 关键字实现的。

方法重写

通过为已声明方法提供一个新实现，可以让派生类重写基类的方法。Java 和 C# 的一个重要区别是：默认情况下，Java 方法被标记为 `virtual`，而在 C# 中，必须使用 `virtual` 修饰符将方法显式标记为 `virtual`。属性访问器和方法均可以重写，它们的重写方法非常相似。

虚拟方法

派生类中要重写的方法使用 `virtual` 修饰符进行声明。在派生类中，已重写的方法是使用 `override` 修饰符声明的。

`override` 修饰符表示派生类的一个方法或属性，它代替基类中具有相同名称和签名的方法或属性。要重写的基方法必须声明为 `virtual`、`abstract` 或 `override`；不能按此方式重写非虚方法或非静态方法。已重写和正在重写的方法或属性必须具有同样的访问级别修饰符。

下面的示例演示派生类中使用重写修饰符进行重写的 `StepUp` 虚拟方法：

C#

```
public class CountClass
{
```

```

public int count;

public CountClass(int startValue) // constructor
{
    count = startValue;
}

public virtual int StepUp()
{
    return ++count;
}

class Count100Class : CountClass
{
    public Count100Class(int x) : base(x) // constructor
    {

    }

    public override int StepUp()
    {
        return ((base.count) + 100);
    }
}

class TestCounters
{
    static void Main()
    {
        CountClass counter1 = new CountClass(1);
        CountClass counter100 = new Count100Class(1);

        System.Console.WriteLine("Count in base class = {0}", counter1.StepUp());
        System.Console.WriteLine("Count in derived class = {0}", counter100.StepUp());
    }
}

```

运行此代码时，可以看到派生类的构造函数使用的是基类中给定的方法体，因此您可以初始化计数成员，而无需复制此代码。输出如下：

```

Count in base class = 2
Count in derived class = 101

```

抽象类

抽象类将一个或多个方法或属性声明为抽象。此类方法不具有声明它们的类中所提供的实现，尽管抽象类也可以包含非抽象方法（即已为其提供了实现的方法）。抽象类不能直接实例化，它只能作为一个派生类。此类派生类必须使用 **override** 关键字为所有抽象方法和属性提供实现，除非派生成员本身被声明为抽象。

下面的示例声明 `Employee` 抽象类。您还将创建一个名为 `Manager` 的派生类，它提供在 `Employee` 类中定义的 `Show()` 抽象方法的实现：

C#

```

public abstract class Employee
{
    protected string name;

    public Employee(string name) // constructor
    {
        this.name = name;
    }

    public abstract void Show(); // abstract show method
}

public class Manager : Employee

```

```

{
    public Manager(string name) : base(name) {} // constructor

    public override void Show() //override the abstract show method
    {
        System.Console.WriteLine("Name : " + name);
    }
}

class TestEmployeeAndManager
{
    static void Main()
    {
        // Create an instance of Manager and assign it to a Manager reference:
        Manager m1 = new Manager("H. Ackerman");
        m1.Show();

        // Create an instance of Manager and assign it to an Employee reference:
        Employee ee1 = new Manager("M. Knott");
        ee1.Show(); //call the show method of the Manager class
    }
}

```

此代码调用 Manager 类提供的 Show() 的实现，并在屏幕上显示员工姓名。输出如下：

```

Name : H. Ackerman
Name : M. Knott

```

接口

接口是一种主干类，它包含方法签名但不包含方法实现。因此，接口类似于只包含抽象方法的抽象类。C# 接口与 Java 接口极其相似，而且工作方式也几乎相同。

根据定义，所有接口成员均为公共成员，并且接口不能包含常量、字段(私有数据成员)、构造函数、析构函数或任何类型的静态成员。如果为某个接口成员指定了任何修饰符，编译器将产生一个错误。

您可以从接口派生类，以实现此接口。此类派生类必须为接口的所有方法提供实现，除非派生类被声明为抽象。

接口声明方式与 Java 中的声明方式相同。在接口定义中，属性仅指示它的类型，并仅通过 `get` 和 `set` 关键字指示它为只读、只写还是读/写。以下接口声明一个只读属性：

C#

```

public interface ICDPlayer
{
    void Play(); // method signature
    void Stop(); // method signature

    int FastForward(float numberSeconds);

    int CurrentTrack // read-only property
    {
        get;
    }
}

```

如果使用冒号代替 Java 中的 **implements** 关键字，则类可以从此接口中继承。实现类必须按如下方式为所有方法及任何所需的属性访问器提供定义：

C#

```

public class CDPlayer : ICDPlayer
{
    private int currentTrack = 0;

    // implement methods defined in the interface
    public void Play()

```

```
{  
    // code to start CD...  
}  
  
public void Stop()  
{  
    // code to stop CD...  
}  
  
public int FastForward(float numberOfSeconds)  
{  
    // code to fast forward CD using numberOfSeconds...  
  
    return 0; //return success code  
}  
  
public int CurrentTrack // read-only property  
{  
    get  
    {  
        return currentTrack;  
    }  
}  
  
// Add additional methods if required...  
}
```

实现多重接口

使用以下语法，可以让类实现多重接口：

C#

```
public class CDAndDVDComboPlayer : ICDPlayer, IDVDPlayer
```

如果一个类在成员名称存在多义性的情况下实现多个接口，则使用属性或方法名称的完全限定符来解析此类。换言之，派生类可以使用方法的完全限定名称表明此派生类属于哪个接口，从而解决冲突，这与 `ICDPlayer.Play()` 中一样。

请参见

参考

[继承 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 编程语言 \(针对 Java 开发人员\)](#)

事件(C# 与 Java)

事件是当对象发生用户关心的情况(如单击图形用户界面上的控件)时,类将此对象通知用户的方法。这种通知即称为“引发事件”。引发事件的对象被称为事件的源或发送方。

与 Java 中的事件处理(通过实现自定义侦听器类执行)不同,C# 开发人员可以使用委托进行事件处理。委托是一种引用方法的类型。一旦为委托分配了方法,委托将与该方法具有完全相同的行为。它与 C++ 函数指针类似,但它是类型安全的。

委托方法的使用方式与任何其他方法相同,都具有参数和返回值,如此示例所示:

```
public delegate int ReturnResult(int x, int y);
```

有关委托的更多信息,请参见[委托\(C# 编程指南\)](#)。

事件和方法一样具有签名,签名包括名称和参数列表。此签名由[委托](#)类型定义,例如:

```
public delegate void MyEventHandler(object sender, System.EventArgs e);
```

通常使用第一个参数作为引用事件的源的对象,使用第二个参数作为携带事件相关数据的对象。但是,在 C# 语言中并不强制使用这种形式;只要事件签名返回 void,其他方面可以与任何有效的委托签名一样。

事件可以使用 **event** 关键字进行声明,如下例所示:

```
public event MyEventHandler TriggerIt;
```

若要触发该事件,请定义引发该事件时要调用的方法,如下例所示:

```
public void Trigger()
{
    TriggerIt();
}
```

若要引发事件,可以调用委托,并传递与该事件相关的参数。然后,该委托将调用已添加到该事件中的所有处理程序。每个事件都可以分配多个处理程序来接收该事件。在这种情况下,该事件将自动调用每个接收方。无论有多少个接收方,引发一个事件都只需调用一次该事件。

如果希望某个类接收事件,请通过使用 **+=** 运算符向该事件添加委托来订阅该事件,例如:

```
myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod);
```

若要取消预订事件,请使用 **-=** 运算符将委托从事件中移除,例如:

```
myEvent.TriggerIt -= new MyEventHandler(myEvent.MyMethod);
```

有关事件的更多信息,请参见[事件\(C# 编程指南\)](#)。

注意

在 C# 2.0 中,委托既可以封装命名方法,也可以封装匿名方法。有关匿名方法的更多信息,请参见[匿名方法\(C# 编程指南\)](#)。

示例

下面的示例定义了一个具有三个关联方法的事件。触发此事件时,将执行这些方法。接着,从此事件移除一个方法,再次触发此事件。

```
// Declare the delegate handler for the event:
public delegate void MyEventHandler();

class TestEvent
```

```

{
    // Declare the event implemented by MyEventHandler.
    public event MyEventHandler TriggerIt;

    // Declare a method that triggers the event:
    public void Trigger()
    {
        TriggerIt();
    }
    // Declare the methods that will be associated with the TriggerIt event.
    public void MyMethod1()
    {
        System.Console.WriteLine("Hello!");
    }
    public void MyMethod2()
    {
        System.Console.WriteLine("Hello again!");
    }
    public void MyMethod3()
    {
        System.Console.WriteLine("Good-bye!");
    }

    static void Main()
    {
        // Create an instance of the TestEvent class.
        TestEvent myEvent = new TestEvent();

        // Subscribe to the event by associating the handlers with the events:
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod1);
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod2);
        myEvent.TriggerIt += new MyEventHandler(myEvent.MyMethod3);
        // Trigger the event:
        myEvent.Trigger();

        // Unsubscribe from the event by removing the handler from the event:
        myEvent.TriggerIt -= new MyEventHandler(myEvent.MyMethod2);
        System.Console.WriteLine("\"Hello again!\" unsubscribed from the event.");

        // Trigger the new event:
        myEvent.Trigger();
    }
}

```

输出

```

Hello!
Hello again!
Good-bye!
"Hello again!" unsubscribed from the event.
Hello!
Good-bye!

```

请参见

参考

[event\(C# 参考\)](#)

[委托\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C#\(针对 Java 开发人员\)](#)

运算符重载(C# 与 Java)

与 C++ 一样, C# 也允许您重载运算符, 以供您自己的类使用。这样做, 可以使使用用户定义的数据类型就像使用基本数据类型一样自然、合理。例如, 您可以创建一个名为 `ComplexNumber` 的新数据类型来表示一个复杂的数字, 并提供使用标准算术运算符对此类数字执行数学运算的方法, 如使用 `+` 运算符将两个复杂数字相加。

若要重载某个运算符, 可以编写一个函数, 在其命名运算符之后加上要重载的运算符的符号。例如, 可按以下方法重载 `+` 运算符:

C#

```
public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
```

所有运算符重载均为类的静态方法。此外还应注意, 重载相等运算符 (`==`) 时, 还必须重载不相等运算符 (`!=`)。`<` 和 `>` 运算符以及 `<=` 和 `>=` 运算符也必须成对重载。

以下是可重载的运算符的完整列表:

- 一元运算符:`+`、`-`、`!`、`~`、`++`、`--`、`true`、`false`
- 二元运算符:`+`、`-`、`*`、`/`、`%`、`&`、`|`、`^`、`<<`、`>>`、`==`、`!=`、`>`、`<`、`>=`、`<=`

下面的代码示例创建一个重载 `+` 和 `-` 运算符的 `ComplexNumber` 类:

C#

```
public class ComplexNumber
{
    private int real;
    private int imaginary;

    public ComplexNumber() : this(0, 0) // constructor
    {
    }

    public ComplexNumber(int r, int i) // constructor
    {
        real = r;
        imaginary = i;
    }

    // Override ToString() to display a complex number in the traditional format:
    public override string ToString()
    {
        return(System.String.Format("{0} + {1}i", real, imaginary));
    }

    // Overloading '+' operator:
    public static ComplexNumber operator+(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real + b.real, a.imaginary + b.imaginary);
    }

    // Overloading '-' operator:
    public static ComplexNumber operator-(ComplexNumber a, ComplexNumber b)
    {
        return new ComplexNumber(a.real - b.real, a.imaginary - b.imaginary);
    }
}
```

使用这个类, 您就可以用以下代码创建和操作两个复杂数字:

C#

```
class TestComplexNumber
{
    static void Main()
    {
        ComplexNumber a = new ComplexNumber(10, 12);
        ComplexNumber b = new ComplexNumber(8, 9);

        System.Console.WriteLine("Complex Number a = {0}", a.ToString());
        System.Console.WriteLine("Complex Number b = {0}", b.ToString());

        ComplexNumber sum = a + b;
        System.Console.WriteLine("Complex Number sum = {0}", sum.ToString());

        ComplexNumber difference = a - b;
        System.Console.WriteLine("Complex Number difference = {0}", difference.ToString());
    }
}
```

如程序所示，您现在可以用非常直观的方式对属于 `ComplexNumber` 类的对象使用加减运算符。结果如下：

```
Complex Number a = 10 + 12i
Complex Number b = 8 + 9i
Complex Number sum = 18 + 21i
Complex Number difference = 2 + 3i
```

尽管 Java 会在内部重载 `+` 运算符以将字符串串联起来，但它并不支持运算符重载。

[请参见](#)

[任务](#)

[“运算符重载”示例](#)

[参考](#)

[可重载运算符 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 编程语言 \(针对 Java 开发人员\)](#)

异常 (C# 与 Java)

C# 中的异常处理与 Java 中的异常处理非常相似。

只要在程序执行过程中出现错误, .NET Framework 公共语言运行库 (CLR) 就会创建一个 [Exception](#) 对象详细描述此错误。在 .NET Framework 中, **Exception** 为所有异常类的基类。从 **Exception** 类派生的异常分为两种类别:[SystemException](#) 和 [ApplicationException](#)。[System](#) 命名空间中的所有类型从 **SystemException** 派生, 而用户定义的异常应从 **ApplicationException** 派生, 以便区分运算时错误和应用程序错误。一些常见的 **System** 异常包括:

- [IndexOutOfRangeException](#): 使用了大于数组或集合大小的索引。
- [NullReferenceException](#): 在将引用设置为有效实例之前使用了引用的属性或方法。
- [ArithemticException](#): 操作将导致上溢或下溢。
- [FormatException](#): 参数或操作数格式不正确。

与 Java 一样, 如果代码可能导致异常, 则将此代码置于 **try** 块内。紧跟在该块之后的一个或多个 **catch** 块提供错误处理。此外还可以对不管是否会引发异常都要执行的任何代码使用 **finally** 块。有关更多信息, 请参见 [try-catch \(C# 参考\)](#) 和 [try-catch-finally \(C# 参考\)](#)。

使用多个 **catch** 块时, 捕捉到的异常必须按照普遍性递增的顺序放置, 因为只有与引发的异常相匹配的第一个 **catch** 块将被执行。C# 编译器对此作了强制规定, 而 Java 编译器则未作强制规定。

此外, C# 不像 Java 那样需要对 **catch** 块使用参数; 在没有参数的情况下, **catch** 块也适用于 **Exception** 类。

例如, 读取文件时可能遇到 [FileNotFoundException](#) 或 [IOException](#) 异常, 并且您希望将含义更明确的 **FileNotFoundException** 处理程序放置在第一位, 如以下代码所示:

C#

```
try
{
    // code to open and read a file
}
catch (System.IO.FileNotFoundException e)
{
    // handle the file not found exception first
}
catch (System.IO.IOException e)
{
    // handle any other IO exceptions second
}
catch
{
    // a catch block without a parameter
    // handle all other exceptions last
}
finally
{
    // this is executed whether or not an exception occurs
    // use to release any external resources
}
```

通过从 **Exception** 派生, 可以创建您自己的异常类。例如, 下面的代码创建一个可能引发的 [InvalidDepartmentException](#) 类 (例如, 如果为新的 [Employee](#) 提供的部门无效)。用户定义的异常的类构造函数使用 **base** 关键字来调用基类构造函数, 并发送相应的消息:

C#

```
public class InvalidDepartmentException : System.Exception
{
    public InvalidDepartmentException(string department) : base("Invalid Department: " + department)
    {
    }
```

```
}
```

接着可以使用以下代码引发一个异常：

C#

```
class Employee
{
    private string department;

    public Employee(string department)
    {
        if (department == "Sales" || department == "Marketing")
        {
            this.department = department;
        }
        else
        {
            throw new InvalidDepartmentException(department);
        }
    }
}
```

C# 不支持选中的异常。在 Java 中，这些选中的异常使用 **throws** 关键字进行声明，从而指定方法可以引发必须由调用代码处理的特定类型异常。

[请参见](#)

[参考](#)

[异常和异常处理\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

高级 C# 技术(C# 与 Java)

C# 提供了一些有用的语言功能(如索引器、属性和委托), 这些功能支持高级编程技术。

索引器

索引器提供了一种像访问数组一样访问类或结构的方法。例如, 可以有一个表示公司中的单个部门的类。这个类可能包含部门中所有雇员的姓名, 而且索引器允许您访问这些姓名, 如下所示:

C#

```
sales[0] = "Nikki";
sales[1] = "Becky";
```

例如, 在类定义中, 通过用以下签名定义属性来启用索引器:

C#

```
public string this [int index] //indexer
```

然后, 像对普通属性一样, 为它提供 `get` 和 `set` 方法, 这些访问器指定当使用该索引器时将引用到什么内部成员。

在下面的简单示例中, 创建一个名为 `Department` 的类, 这个类使用索引器访问该部门中的雇员, 这些雇员在内部表示为一个字符串数组:

C#

```
public class Department
{
    private string name;
    private const int MAX_EMPLOYEES = 10;
    private string[] employees = new string[MAX_EMPLOYEES]; //employee array

    public Department(string departmentName) //constructor
    {
        name = departmentName;
    }

    public string this [int index] //indexer
    {
        get
        {
            if (index >= 0 && index < MAX_EMPLOYEES)
            {
                return employees[index];
            }
            else
            {
                throw new System.IndexOutOfRangeException();
            }
        }
        set
        {
            if (index >= 0 && index < MAX_EMPLOYEES)
            {
                employees[index] = value;
            }
            else
            {
                throw new System.IndexOutOfRangeException();
            }
        }
    }
}
```

```
// code for the rest of the class...
}
```

然后，可以创建这个类的一个实例并访问它，如下面的代码示例所示：

C#

```
class TestDepartment
{
    static void Main()
    {
        Department sales = new Department("Sales");

        sales[0] = "Nikki";
        sales[1] = "Becky";

        System.Console.WriteLine("The sales team is {0} and {1}", sales[0], sales[1]);
    }
}
```

输出为：

```
The sales team is Nikki and Becky
```

有关更多信息，请参见索引器(C# 编程指南)。

属性

C# 提供了一种称为“属性”的机制，用于添加有关类型的声明性信息。属性有些类似于 Java 中的批注概念。关于类型的额外信息位于类型定义前面的声明标记中。下面的示例演示如何使用 .NET Framework 属性来修饰类或方法。

在下面的示例中，通过添加 **WebMethodAttribute** 属性将 `GetTime` 方法标记为 XML Web services。

C#

```
public class Utilities : System.Web.Services.WebService
{
    [System.Web.Services.WebMethod] // Attribute
    public string GetTime()
    {
        return System.DateTime.Now.ToShortTimeString();
    }
}
```

添加 **WebMethod** 属性使 .NET Framework 自动处理调用此函数所需的 XML/SOAP 交换。调用此 Web 服务将检索下面的值：

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">7:26 PM</string>
```

在下面的示例中，通过添加 **SerializableAttribute** 属性将 `Employee` 类标记为 `Serializable`。虽然 `Salary` 字段被标记为 `public`，但是它不会被序列化，因为它标记了 `NonSerializedAttribute` 属性。

C#

```
[System.Serializable()]
public class Employee
{
    public int ID;
    public string Name;
    [System.NonSerialized()] public int Salary;
}
```

有关更多信息，请参见[创建自定义属性\(C# 编程指南\)](#)。

委托

C++、Pascal 和其他语言支持函数指针的概念，允许您在运行时选择要调用哪些函数。

Java 不提供任何具有函数指针功能的结构，但 C# 提供这种构造。通过使用 [Delegate](#) 类，[委托](#) 实例可以封装属于可调用实体的方法。

对于实例方法，委托由一个包含类的实例和该实例上的方法组成。对于静态方法，可调用实体由一个类和该类上的静态方法组成。因此，委托可用于调用任何对象的函数，而且委托是面向对象的、类型安全和安全的。

定义和使用委托有三个步骤：

- 声明
- 实例化
- 调用

使用以下语法声明委托：

C#

```
delegate void Del1();
```

然后可以使用此委托引用返回 void 而且不带任何参数的函数。

类似地，若要为任何带字符串参数且返回 long 型结果的函数创建委托，则使用以下语法：

C#

```
delegate long Del2(string s);
```

然后，可以将此委托分配到带有此签名的任何方法，如下所示：

C#

```
Del2 d; // declare the delegate variable  
d = DoWork; // set the delegate to refer to the DoWork method
```

其中 DoWork 的签名为：

C#

```
public static long DoWork(string name)
```

重新分配委托

[Delegate](#) 对象是不可变的，即设置与它们匹配的签名后就不能再更改签名了。但是，如果其他方法具有同一签名，您也可以指向该方法。在本例中，将 d 重新分配给一个新的委托对象，因此 d 将调用 DoMoreWork 方法。只有 DoWork 和 DoMoreWork 具有相同签名时，您才可以执行此操作。

C#

```
Del2 d; // declare the delegate variable  
d = DoWork; // set the delegate to refer to the DoWork method  
d = DoMoreWork; // reassign the delegate to refer to the DoMoreWork method
```

调用委托

调用委托相当简单。只需用委托变量的名称替代方法名称。这将使用值 11 和 22 调用 Add 方法，并返回一个 long 型结果，该结果被赋给变量 sum：

C#

```
Del operation; // declare the delegate variable
```

```
operation = Add;      // set the delegate to refer to the Add method
long sum = operation(11, 22); // invoke the delegate
```

以下代码演示了委托的创建、实例化和调用：

C#

```
public class MathClass
{
    public static long Add(int i, int j)      // static
    {
        return (i + j);
    }

    public static long Multiply (int i, int j) // static
    {
        return (i * j);
    }
}

class TestMathClass
{
    delegate long Del(int i, int j); // declare the delegate type

    static void Main()
    {
        Del operation; // declare the delegate variable

        operation = MathClass.Add;      // set the delegate to refer to the Add method
        long sum = operation(11, 22);      // use the delegate to call the Add method

        operation = MathClass.Multiply; // change the delegate to refer to the Multiply method
        long product = operation(30, 40);      // use the delegate to call the Multiply method

        System.Console.WriteLine("11 + 22 = " + sum);
        System.Console.WriteLine("30 * 40 = " + product);
    }
}
```

输出

```
11 + 22 = 33
30 * 40 = 1200
```

委托实例必须包含一个对象引用。前面的示例通过将方法声明为静态（意味着无需指定对象引用）避开了这一要求。但如果委托指实例方法，则必须按如下方式给定对象引用：

C#

```
Del operation;          // declare the delegate variable
MathClass m1 = new MathClass(); // declare the MathClass instance
operation = m1.Add;      // set the delegate to refer to the Add method
```

在本例中，Add 和 Multiply 是 MathClass 的实例方法。如果 MathClass 的方法未声明为静态方法，那么您可以使用 MathClass 的实例通过委托来调用这些方法，如下所示：

C#

```
public class MathClass
{
    public long Add(int i, int j)      // not static
    {
```

```

        return (i + j);
    }

    public long Multiply (int i, int j) // not static
    {
        return (i * j);
    }
}

class TestMathClass
{
    delegate long Del(int i, int j); // declare the delegate type

    static void Main()
    {
        Del operation; // declare the delegate variable
        MathClass m1 = new MathClass(); // declare the MathClass instance

        operation = m1.Add; // set the delegate to refer to the Add method
        long sum = operation(11, 22); // use the delegate to call the Add method

        operation = m1.Multiply; // change the delegate to refer to the Multiply method
        long product = operation(30, 40); // use the delegate to call the Multiply method

        System.Console.WriteLine("11 + 22 = " + sum);
        System.Console.WriteLine("30 * 40 = " + product);
    }
}

```

输出

本示例提供的输出与之前方法声明为静态的示例相同。

```

11 + 22 = 33
30 * 40 = 1200

```

委托与事件

.NET Framework 还可以广泛地将委托用于事件处理任务，如 Windows 或 Web 应用程序中的按钮 Click 事件。Java 中的事件处理通常通过实现自定义侦听器类完成，而 C# 开发人员则可以利用委托处理事件。[事件](#)的声明类似于具有委托类型的字段，区别在于事件声明前面有 event 关键字。事件通常被声明为 [public](#)，但允许使用任何可访问性修饰符。下面的示例演示了 **delegate** 和 **event** 的声明。

C#

```

// Declare the delegate type:
public delegate void CustomEventHandler(object sender, System.EventArgs e);

// Declare the event variable using the delegate type:
public event CustomEventHandler CustomEvent;

```

事件委托是多路广播的，这意味着它们可以对多个事件处理方法进行引用。通过维护事件的已注册事件处理程序列表，委托为引发事件的类担当事件发送器的角色。下面的示例演示如何为多个函数订阅事件。EventClass 类包含委托、事件和调用事件的方法。请注意调用事件只能从声明该事件的类内部进行。然后，TestEvents 类使用 += 运算符订阅事件，并使用 -= 运算符取消订阅。调用 InvokeEvent 方法时，它将激发事件，所有订阅了该事件的函数也同步激发，如下面的示例所示。

C#

```

public class EventClass
{
    // Declare the delegate type:
    public delegate void CustomEventHandler(object sender, System.EventArgs e);

    // Declare the event variable using the delegate type:
    public event CustomEventHandler CustomEvent;

```

```

public void InvokeEvent()
{
    // Invoke the event from within the class that declared the event:
    CustomEvent(this, System.EventArgs.Empty);
}

class TestEvents
{
    private static void CodeToRun(object sender, System.EventArgs e)
    {
        System.Console.WriteLine("CodeToRun is executing");
    }

    private static void MoreCodeToRun(object sender, System.EventArgs e)
    {
        System.Console.WriteLine("MoreCodeToRun is executing");
    }

    static void Main()
    {
        EventClass ec = new EventClass();

        ec.CustomEvent += new EventClass.CustomEventHandler(CodeToRun);
        ec.CustomEvent += new EventClass.CustomEventHandler(MoreCodeToRun);

        System.Console.WriteLine("First Invocation:");
        ec.InvokeEvent();

        ec.CustomEvent -= new EventClass.CustomEventHandler(MoreCodeToRun);

        System.Console.WriteLine("\nSecond Invocation:");
        ec.InvokeEvent();
    }
}

```

输出

```

First Invocation:
CodeToRun is executing
MoreCodeToRun is executing
Second Invocation:
CodeToRun is executing

```

请参见

[任务](#)

[“委托”示例](#)

[概念](#)

[C# 编程指南](#)

[委托 \(C# 编程指南\)](#)

[事件 \(C# 编程指南\)](#)

[其他资源](#)

[C# 编程语言 \(针对 Java 开发人员\)](#)

垃圾回收 (C# 与 Java)

在 C 和 C++ 中，许多对象要求程序员在声明它们后为其分配资源，然后才可以安全地使用对象。使用完对象后将这些资源释放回自由内存池，也是程序员的责任。如果资源得不到释放，则认为代码泄漏内存，因为越来越多的资源会被不必要地消耗掉。另一方面，如果资源被过早释放，可能会发生数据丢失、其他内存区域损坏和 Null 指针异常。

Java 和 C# 都单独管理应用程序使用的所有对象的生存期，通过这种方法防范这些危险。

在 Java 中，JVM 通过跟踪对已分配资源的引用来自动释放不再使用的内存。只要 JVM 检测到不再由有效引用引用的资源，该资源就被作为垃圾回收。

在 C# 中，垃圾回收由与 JVM 的功能类似的公共语言运行库 (CLR) 处理。CLR 垃圾回收器定期检查内存堆中是否有任何未引用的对象，并释放这些对象占用的资源。

请参见

概念

[C# 编程指南](#)

[自动内存管理](#)

[其他资源](#)

[C# 代码示例\(针对 Java 开发人员\)](#)

安全代码与不安全的代码 (C# 与 Java)

C# 的一项特别有趣的功能是支持不安全类型的代码。通常，公共语言运行库 (CLR) 负责检查 Microsoft 中间语言 (MSIL) 代码的行为，防止任何有问题的操作。但是，有时您希望直接访问低级功能（如 Win32 API 调用），只要您负责确保此类代码能够正确运行，就允许您这样做。此类代码必须放在源代码中的不安全块内。

Unsafe 关键字

进行低级 API 调用、使用指针算法或执行其他一些棘手操作的 C# 代码必须放在以 `unsafe` 关键字标记的块内。下面的任何代码均可标记为 `unsafe`：

- 整个方法。
- 大括号中的代码块。
- 单个语句。

下面的示例演示如何在上述三种情况下使用 `unsafe`：

C#

```
class TestUnsafe
{
    unsafe static void PointyMethod()
    {
        int i=10;

        int *p = &i;
        System.Console.WriteLine("*p = " + *p);
        System.Console.WriteLine("Address of p = {0:X2}\n", (int)p);
    }

    static void StillPointy()
    {
        int i=10;

        unsafe
        {
            int *p = &i;
            System.Console.WriteLine("*p = " + *p);
            System.Console.WriteLine("Address of p = {0:X2}\n", (int)p);
        }
    }

    static void Main()
    {
        PointyMethod();
        StillPointy();
    }
}
```

在这段代码中，整个 `PointyMethod()` 方法被标记为 `unsafe`，因为该方法声明和使用了指针。`StillPointy()` 方法将一个代码块标记为 `unsafe`，因为这个代码块再次使用了指针。

fixed 关键字

在安全代码中，垃圾回收器在对象的生存期内可以自由地移动对象，以组织和压缩可用资源。但是，如果代码使用了指针，则此行为可能很容易造成意外的结果，因此您可以使用 `fixed` 语句来指示垃圾回收器不要移动某些对象。

下面的代码演示了使用 `fixed` 关键字以确保在执行 `PointyMethod()` 方法中的代码块时系统不会移动数组。注意：`fixed` 只用于不安全的代码中：

C#

```
class TestFixed
{
```

```
public static void PointyMethod(char[] array)
{
    unsafe
    {
        fixed (char *p = array)
        {
            for (int i=0; i<array.Length; i++)
            {
                System.Console.Write(*(p+i));
            }
        }
    }

    static void Main()
    {
        char[] array = { 'H', 'e', 'l', 'l', 'o' };
        PointyMethod(array);
    }
}
```

[请参见](#)

[任务](#)

[“不安全代码”示例](#)

[概念](#)

[C# 编程指南](#)

[不安全代码和指针 \(C# 编程指南\)](#)

[其他资源](#)

[C# 编程语言 \(针对 Java 开发人员\)](#)

C# 代码示例(针对 Java 开发人员)

C# 是一种精确、简单、类型安全、面向对象的语言，它使程序员可以构建各种各样的应用程序。与 .NET Framework 组合后，Visual C# 使用户可以创建 Windows 应用程序、Web 服务、数据库工具、组件、控件以及其他内容。

本节内容

[控制台应用程序开发\(C# 与 Java\)](#)

描述 C# 中的控制台应用程序(针对 Java 开发人员)。

[文件 I/O\(C# 与 Java\)](#)

描述 C# 中针对 Java 开发人员的文件 I/O 操作，并链接到 XML 类以供 XML 文件进行加载和保存。

[数据库访问\(C# 与 Java\)](#)

比较两种语言中的数据访问。

[用户界面开发\(C# 与 Java\)](#)

描述用 C# 创建 Windows 窗体应用程序(针对 Java 开发人员)。

[资源管理\(C# 与 Java\)](#)

提供关于用 C# 编写的 Windows 资源文件的主题(针对 Java 开发人员)。

[Web 服务应用程序\(C# 与 Java\)](#)

描述用 C# 进行 XML Web 服务应用程序开发(针对 Java 开发人员)。

[移动设备和数据\(C# 与 Java\)](#)

描述使用 C#、.NET Framework Windows 窗体和 SQL Server 与数据库交互(针对 Java 开发人员)。

请参见

概念

[C# 编程指南](#)

其他资源

[迁移到 Visual C#](#)

[用 Visual C# 编写应用程序](#)

控制台应用程序开发 (C# 与 Java)

控制台应用程序无需使用任何图形用户界面即可读取标准输入和输出 (I/O) 中的数据并向其写入数据。Java 和 C# 的控制台应用程序结构十分相似，而且控制台 I/O 所使用的类也十分相似。

尽管类和方法签名的详细信息可能不同，但 C# 和 Java 都使用类似的概念来执行控制台 I/O 操作。对于控制台应用程序及相关的控制台读取和写入方法，C# 和 Java 均使用主入口点这一概念。在 C# 中，主入口点为 `Main`，而在 Java 中，主入口点为 `main`。

Java“Hello World”示例

在下面的 Java 示例代码中，`static void main()` 例程接受一个对应用程序参数的 `String` 引用，然后 `main` 例程将一行输出到控制台。

```
/* A Java Hello World Console Application */
public class Hello {
    public static void main (String args[]) {
        System.out.println ("Hello World");
    }
}
```

C#“Hello World”示例

在下面的 C# 示例代码中，`static void Main()` 例程接受一个对应用程序参数的 `string` 引用，然后 `Main` 例程将一行代码写入控制台。

C#

```
// A C# Hello World Console Application.
public class Hello
{
    static void Main()
    {
        System.Console.WriteLine("Hello World");
    }
}
```

编译代码

如果您使用的是 Visual C#，按 F5 可以一步编译并运行代码。如果您使用的是命令行并且文件名为“Hello.cs”，则可以如下所示调用 C# 编译器：

csc Hello.cs

更多信息

有关创建控制台应用程序的更多信息，请参见[创建控制台应用程序 \(Visual C#\)](#)。

有关 .NET Framework 控制台类的更多信息，请参见：

- [Console 类、`WriteLine` 方法和 `ReadLine` 方法。](#)
- [按类别列出的 C# 编译器选项](#) .
- [C# 2.0 语言和编译器中的新增功能](#)

有关从 Java 到 C# 的自动转换的更多信息，请参见

[What's New in Java Language Conversion Assistant \(Java Language Conversion Assistant 中的新增功能\)](#)。

请参见

参考

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C#\(针对 Java 开发人员\)](#)

文件 I/O(C# 与 Java)

尽管类和方法签名的详细信息可能不同，但 C# 和 Java 在执行 I/O 操作时使用的概念类似。C# 和 Java 均有 file 类的概念和关联的文件读写方法。存在相似的文档对象模型 (DOM)，用于处理 XML 内容。

Java 文件操作示例

在 Java 中，可以使用 **File** 对象执行基本的文件 I/O 操作，例如创建、打开、关闭、读取和写入文件。例如，可以使用 **File** 类的方法执行文件 I/O 操作，例如使用 **File** 类的 **createNewFile** 或 **delete** 方法创建或删除文件。可以使用 **BufferedReader** 和 **BufferedWriter** 类读写文件的内容。

下面的代码示例演示如何创建新文件、删除文件、从文件中读取文本以及写入文件。

```
// Java example code to create a new file
try
{
    File file = new File("path and file_name");
    boolean success = file.createNewFile();
}
catch (IOException e)      {      }

// Java example code to delete a file.
try
{
    File file = new File("path and file_name");
    boolean success = file.delete();
}
catch (IOException e)      {      }

// Java example code to read text from a file.
try
{
    BufferedReader infile = new BufferedReader(new FileReader("path and file_name "));
    String str;
    while ((str = in.readLine()) != null)
    {
        process(str);
    }
    infile.close();
}
catch (IOException e)
{
    // Exceptions ignored.
}

// Java example code to writing to a file.
try
{
    BufferedWriter outfile =
        new BufferedWriter(new FileWriter("path and file_name "));
    outfile.write("a string");
    outfile.close();
}
catch (IOException e)      {      }
```

C# 文件操作示例

要在 C# 中执行一个文件 I/O 操作，可以使用同样熟悉的基本步骤，使用 .NET Framework 等效类和方法执行创建、打开、关闭、读取和写入操作。例如，可以使用 .NET Framework 的 **File** 类的方法执行文件 I/O 操作。例如，可以使用 **Exists** 方法检查文件是否存在。可以使用 **Create** 方法来创建文件，在此过程中可以选择改写现有文件（如下面的代码示例所示），并且可以使用 **FileStream** 类和 **BufferedStream** 对象来执行读写操作。

下面的代码示例演示如何删除文件、创建文件、写入文件和读取文件。

```

// sample C# code for basic file I/O operations
// exceptions ignored for code simplicity

class TestFileIO
{
    static void Main()
    {
        string fileName = "test.txt"; // a sample file name

        // Delete the file if it exists.
        if (System.IO.File.Exists(fileName))
        {
            System.IO.File.Delete(fileName);
        }

        // Create the file.
        using (System.IO.FileStream fs = System.IO.File.Create(fileName, 1024))
        {
            // Add some information to the file.
            byte[] info = new System.Text.UTF8Encoding(true).GetBytes("This is some text in
the file.");
            fs.Write(info, 0, info.Length);
        }

        // Open the file and read it back.
        using (System.IO.StreamReader sr = System.IO.File.OpenText(fileName))
        {
            string s = "";
            while ((s = sr.ReadLine()) != null)
            {
                System.Console.WriteLine(s);
            }
        }
    }
}

```

相关章节

可用于创建、读取和写入流的 .NET Framework 类包括 [StreamReader](#) 和 [StreamWriter](#) 类。可用于处理文件的其他 .NET Framework 类包括：

- [FileAccess](#) 和 [FileAttribute](#) 类。
- [Directory](#) 和 [DirectoryInfo](#)、[Path](#)、 [FileInfo](#) 和 [DriveInfo](#) 类。
- [BinaryReader](#) 和 [BinaryWriter](#) 类。
- [StringReader](#) 和 [StringWriter](#) 类。
- [TextReader](#) 和 [TextWriter](#) 类
- [XmlReader](#) 和 [XmlWriter](#) 类。
- [ToBase64Transform](#) 类。
- **FileStream**、**BufferedStream** 和 [MemoryStream](#) 类。

有关从 Java 到 C# 的自动转换的更多信息，请参见 [Java Language Conversion Assistant 3.0 中的新增功能](#)。

有关 .NET Framework 安全性的更多信息，请参见 [NET Security \(NET 安全性\)](#)。

请参见

概念

[C# 编程指南](#)

[异步文件 I/O](#)

其他资源

[C#\(针对 Java 开发人员\)](#)

数据库访问 (C# 与 Java)

C# 和 Java 使用相似的方式访问数据库数据。C# 和 Java 都需要数据库驱动程序来执行实际的数据库操作。另外，两者都需要一个数据库连接、一个要针对数据库连接执行的 SQL 查询以及一个执行查询后产生的结果集。

比较数据库驱动程序

在 Java 和 C# 中，可以使用 JDBC 或 ODBC 等数据库驱动程序来访问数据。Java 数据库连接 (JDBC) 驱动程序用在以 Java 编写的程序中。开放式数据库连接 (ODBC) 是 Microsoft 的数据库编程接口，用于访问许多平台上的各种关系数据库。Solaris 和 Windows 版本的 Java 平台上也都有 JDBC-ODBC 桥接标准，因此您也可以在 Java 程序中使用 ODBC。

在 Java 中，为连接句柄的驱动程序提供了连接字符串信息，如下所示：

```
final static private String url = "jdbc:oracle:server,user,pass, ...";
```

在 C# 中，通过使用 .NET Framework，您不必为了访问数据库而加载 ODBC 或 JDBC 驱动程序。只需为数据库连接对象设置连接字符串，如下所示：

C#

```
static string connectionString = "Initial Catalog=northwind;Data Source=(local);Integrated Security=SSPI;";
static SqlConnection cn = new SqlConnection(connectionString);
```

- 有关 Oracle 数据库的 ODBC 驱动程序的更多信息，请参见“[ODBC Driver for Oracle](#)”(Oracle 的 ODBC 驱动程序)。有关 DB2 数据库的 OLE DB 提供程序的附加信息，请参见“[Microsoft Host Integration Server 2000 Developer's Guide](#)”(Microsoft Host Integration Server 2000 开发人员指南)和“[Administration and Management of Data Access Using the OLE DB Provider for SQL Server 2000](#)”。
- “[Microsoft® SQL Server™ 2000 Driver for JDBC](#)”(JDBC 的 Microsoft® SQL Server™ 2000 驱动程序)是一个 Type 4 JDBC 驱动程序，它提供从任何支持 Java 的 Applet、应用程序或应用程序服务器对 SQL Server 2000 的访问。

Java 读取数据库示例

在 Java 中若要执行数据库读取操作，可以使用由 **Statement** 对象的 **executeQuery** 方法创建的 **ResultSet** 对象。**ResultSet** 对象包含由查询返回的数据。然后您可以循环访问 **ResultSet** 对象以访问此数据。

下面的示例提供的 Java 代码用于读取数据库。

```
Connection c;
try
{
    Class.forName (_driver);
    c = DriverManager.getConnection(url, user, pass);
}
catch (Exception e)
{
    // Handle exceptions for DriverManager
    // and Connection creation:
}
try
{
    Statement stmt = c.createStatement();
    ResultSet results = stmt.executeQuery(
        "SELECT TEXT FROM dba ");
    while(results.next())
    {
        String s = results.getString("ColumnName");
        // Display each ColumnName value in the ResultSet:
    }
    stmt.close();
}
catch(java.sql.SQLException e)
{
    // Handle exceptions for executeQuery and getString:
}
```

同样地，若要执行数据库写入操作，则从 **Connection** 对象创建 **Statement** 对象。**Statement** 对象具有依据数据库执行 SQL 查询和更新的方法。更新和查询的格式为包含写入操作的 SQL 命令的字符串，在 **Statement** 对象的 **executeUpdate** 方法中使用此写入操作以返回 **ResultSet** 对象。

C# 读取数据库示例

在 C# 中，通过使用 .NET Framework，可借助 ADO.NET 提供的类集进一步简化对数据的访问，这将支持使用 ODBC 驱动程序以及通过 OLE DB 提供程序来访问数据库。C# 应用程序可以使用 .NET Framework 的 **ADO.NET** 类和通过 Microsoft 数据访问组件 (MDAC) 与 SQL 数据库交互，以执行读取、写入和搜索数据等操作。.NET Framework 的 **System.Data.SqlClient** 命名空间和类使访问 SQL Server 数据库更加简单。

在 C# 中，可以使用连接、命令和数据表来执行数据库读取操作。例如，若要使用 **System.Data.SqlClient** 命名空间连接到 SQL Server 数据库，可以使用下面的类：

- **SqlConnection** 类。
- 查询，如 **SqlCommand** 类。
- 结果集，如 **DataTable** 类。

.NET Framework 提供 **DataAdapter**，它将以下三个对象汇集在一起，如下所示：

- 使用 **DataAdapter** 对象的连接属性设置 **SqlConnection** 对象。
- 使用 **DataAdapter** 的 **SelectCommand** 属性指定要执行的查询。
- 使用 **DataAdapter** 对象的 **Fill** 方法创建 **DataTable** 对象。**DataTable** 对象包含由查询返回的结果集数据。可以循环访问 **DataTable** 对象，以便使用行集合访问数据行。

若要编译和运行代码，需要满足以下条件；否则，行 `databaseConnection.Open();` 会失败并引发异常。

- Microsoft 数据访问组件 (MDAC) 2.7 或更高版本。

如果您使用的是 Microsoft Windows XP 或 Windows Server 2003，那么您已经有了 MDAC 2.7。但是，如果使用的是 Microsoft Windows 2000，则可能需要升级计算机上已安装的 MDAC。有关更多信息，请参见“[MDAC Installation](#)”(MDAC 安装)。

- 当前用户名访问 SQL Server Northwind 数据库的权限和集成安全特权，此用户名在安装了 Northwind 示例数据库的本地 SQL Server 上运行代码。

C#

```
// Sample C# code accessing a sample database

// You need:
//   A database connection
//   A command to execute
//   A data adapter that understands SQL databases
//   A table to hold the result set

namespace DataAccess
```

```

{
    using System.Data;
    using System.Data.SqlClient;

    class DataAccess
    {
        //This is your database connection:
        static string connectionString = "Initial Catalog=northwind;Data Source=(local);Integrated Security=SSPI";
        static SqlConnection cn = new SqlConnection(connectionString);

        // This is your command to execute:
        static string sCommand = "SELECT TOP 10 Lastname FROM Employees ORDER BY EmployeeID";

        // This is your data adapter that understands SQL databases:
        static SqlDataAdapter da = new SqlDataAdapter(sCommand, cn);

        // This is your table to hold the result set:
        static DataTable dataTable = new DataTable();

        static void Main()
        {
            try
            {
                cn.Open();

                // Fill the data table with select statement's query results:
                int recordsAffected = da.Fill(dataTable);

                if (recordsAffected > 0)
                {
                    foreach (DataRow dr in dataTable.Rows)
                    {
                        System.Console.WriteLine(dr[0]);
                    }
                }
            }
            catch (SqlException e)
            {
                string msg = "";
                for (int i=0; i < e.Errors.Count; i++)
                {
                    msg += "Error #" + i + " Message: " + e.Errors[i].Message + "\n";
                }
                System.Console.WriteLine(msg);
            }
            finally
            {
                if (cn.State != ConnectionState.Closed)
                {
                    cn.Close();
                }
            }
        }
    }
}

```

有关 ADO.NET 的更多信息, 请参见:

- [ADO.NET](#)
- [访问数据 \(Visual Studio\)](#)
- [ADO.NET 示例应用程序](#)
- [ADO.NET for the Java Programmer \(ADO.NET\(针对 Java 程序员\)\)](#)

有关 .NET Framework 数据库访问类的更多信息, 请参见:

- [System.Data.OracleClient](#)
- [System.Data.SqlServerCe](#)
- [System.Data.Odbc](#)
- [System.Data.OleDb](#)
- [ADO.NET DataSet](#)
- [在 ADO.NET 中使用 DataSet](#)

有关 Java Language Conversion Assistant 的更多信息, 请参见 [Java Language Conversion Assistant 3.0 中的新增功能](#)。

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C#\(针对 Java 开发人员\)](#)

用户界面开发(C# 与 Java)

可以在 C# 中使用 .NET Framework 的丰富 Windows 窗体组件进行客户端窗体应用程序编程。

Java

大部分 Java 应用程序使用 Abstract Windowing ToolKit (AWT) 或 Swing(包括 AWT 事件模型)来进行窗体编程, 而 Swing 使用 AWT 的基础结构。AWT 提供所有基本的 GUI 功能和类。

Java 示例

框架是带有标题和边框的窗口, 通常用于添加组件。

```
JFrame aframe = new JFrame();
```

Component 类是具有图形表示形式的对象, 它经常会被扩展; 使用继承的方法也经常重写这些方法, 如演示代码中的 **Shape** 组件的 **paint** 方法。

```
import java.awt.*;
import javax.swing.*;

class aShape extends JComponent {
    public void paint(Graphics g) {
        Graphics2D g2d = (Graphics2D)g;

        // Draw the shape.
    }

    public static void main(String[] args) {
        JFrame aframe = new JFrame();
        frame.getContentPane().add(new aShape ());
        int frameWidth = 300;
        int frameHeight = 300;
        frame.setSize(frameWidth, frameHeight);
        frame.setVisible(true);
    }
}
```

可以注册侦听器来侦听组件的操作事件, 从而处理事件。例如, 按下并释放某个按钮时, AWT 通过调用此按钮上的 **processEvent** 将一个 **ActionEvent** 实例发送到该按钮。此按钮的 **processEvent** 方法接收此按钮的所有事件, 它通过调用自身的 **processActionEvent** 方法来继续传递操作事件。后一个方法将继续传递操作事件, 将它传递给那些已注册为侦听此按钮生成的操作事件的操作侦听器。

C#

在 C# 中, .NET Framework 的 [System.Windows.Forms](#) 命名空间和类为 Windows 窗体开发提供一套全面的组件。例如, 下面的代码使用 [Label](#)、[Button](#) 和 [MenuStrip](#)。

C# 示例

直接从 [Form](#) 类派生, 如下所示:

C#

```
public partial class Form1 : System.Windows.Forms.Form
```

添加您的组件:

C#

```
this.button1 = new System.Windows.Forms.Button();
this.Controls.Add(this.button1);
```

下面的代码演示如何将标签、按钮和菜单添加到窗体。

C#

```
namespace WindowsFormApp
{
    public partial class Form1 : System.Windows.Forms.Form
    {
        private System.ComponentModel.Container components = null;

        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.MenuStrip menu1;

        public Form1()
        {
            InitializeComponent();
        }

        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();

            this.label1 = new System.Windows.Forms.Label();
            this.Controls.Add(this.label1);

            this.button1 = new System.Windows.Forms.Button();
            this.Controls.Add(this.button1);

            this.menu1 = new System.Windows.Forms.MenuStrip();
            this.Controls.Add(this.menu1);
        }

        static void Main()
        {
            System.Windows.Forms.Application.Run(new Form1());
        }
    }
}
```

与 Java 类似，在 C# 中可以注册侦听器来侦听组件的事件。例如，按下并释放按钮时，运行库会将一个 **Click** 事件传递给所有已注册于侦听该按钮的 **Click** 事件的侦听器。

C#

```
private void button1_Click(object sender, System.EventArgs e)
{
}
```

例如，可以使用下面的代码来注册 `button1_Click`，以处理 **Button** 的实例（称为 `button1`）的 **Click** 事件。

C#

```
// this code can go in InitializeComponent()
button1.Click += button1_Click;
```

有关更多信息，请参见[创建 ASP.NET Web 应用程序 \(Visual C#\)](#)。

有关从 Java 到 C# 的自动转换的更多信息，请参见[Java Language Conversion Assistant 3.0 中的新增功能](#)。

有关 **Forms** 类的更多信息，请参见[根据功能列出的 Windows 窗体控件](#)和 **System.Windows.Forms**。

请参见

概念

[C# 编程指南](#)

[设计用户界面 \(Visual C#\)](#)

[其他资源](#)

[C#\(针对 Java 开发人员\)](#)

资源管理(C# 与 Java)

在 C# 中，使用 Visual Studio 管理资源变得更加简单。

Java

Java 应用程序通常与应用程序的各种资源例如类文件、声音文件和图像文件一起绑定在 JAR 文件中。您很可能使用 JBuilder 或 Eclipse，按照与 Visual Studio 管理解决方案和项目很相似的方式来管理 JAR 文件。

C#

在 C# 项目中，可以直接从 Visual Studio 的[解决方案资源管理器](#)中打开资源。

也可以使用[图像编辑器](#)和[二进制编辑器](#)处理托管项目中的资源文件。

有关向托管项目添加资源的更多信息，请参见：

- [添加和编辑资源 \(Visual C#\)](#)
- [演练:本地化 Windows 窗体](#)
- [演练:将本地化资源用于 ASP.NET](#)

可以像读取外部内容或嵌入资源一样读取应用程序中的这些资源。例如，下面的代码行使用 [System.Reflection](#) 命名空间中的类以及 [Assembly](#) 这样的类从程序集读取嵌入的资源文件。此例中的文件为 `assemblyname.file.ext`。

C#

```
static void Main()
{
    System.Reflection.Assembly asm =
        System.Reflection.Assembly.GetExecutingAssembly();

    System.Drawing.Bitmap tiles = new System.Drawing.Bitmap
        (asm.GetManifestResourceStream("assemblyname.file.ext"));
}
```

- 有关更多信息，请参见[反射 \(C# 编程指南\)](#)。
- 有关应用程序资源的更多信息，请参见[管理应用程序资源](#)。
- 有关常规资源编辑器的工作原理信息，请参见[资源编辑器](#)。
- 有关编辑 .Resx 格式的资源文件的更多信息，请参见[应用程序中的资源](#)。
- 有关处理 XML 和 Simplified API for XML (SAX2) 的更多信息，请参见 [SAX2 Developer Guide \(SAX2 开发人员指南\)](#) 和 [XML Developer Center \(XML 开发中心\)](#)。

请参见

概念

[C# 编程指南](#)

[添加和编辑资源 \(Visual C#\)](#)

其他资源

[C#\(针对 Java 开发人员\)](#)

Web 服务应用程序 (C# 与 Java)

.NET Framework 为通过 Web 服务进行互操作提供了广泛的支持。在 C# 中使用 .NET Framework、Visual Studio 和 ASP.NET 创建一个 Web 服务就像创建一个 Web 服务项目并将属性 **WebMethod** 添加到您要公开的任何公用方法一样简单。

Java

在 Java 中您可以使用 Web 服务包来实现应用程序，如 Java Web Services Developer Pack 或 Apache SOAP。例如，在 Java 中可以使用下面的步骤创建一个 Web 服务和 Apache SOAP。

在 Java 中使用 Apache SOAP 创建 Web 服务

1. 如下所示编写一个 Web 服务方法：

```
public class HelloWorld
{
    public String sayHelloWorld()
    {
        return "HelloWorld ";
    }
}
```

2. 创建 Apache SOAP 部署说明符。这可能与如下所示的说明符相似：

```
<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"
    id="urn:HelloWorld">

    <dd:provider type="java"
        scope="Application"
        methods="sayHelloWorld">

        <dd:java class="HelloWorld" static="false" />

    </dd:provider>

    <dd:faultListener>org.apache.soap.server.DOMFaultListener</dd:faultListener>

    <dd:mappings />

</dd:service>
```

3. 编译 HelloWorld 类并将它移到 Web 服务器的类路径。

4. 使用命令行工具部署 Web 服务。

C#

在 C# 中使用 .NET Framework 类和 Visual Studio IDE 创建 Web 服务更为简单。

在 C# 中使用 .NET Framework 和 Visual Studio 创建 Web 服务

1. 在 Visual Studio 中创建 Web 服务应用程序。有关更多信息，请参见 [C# 应用程序类型 \(针对 Java 开发人员\)](#)。生成的代码如下所示。

C#

```
using System;
using System.Web;
```

```

using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service () {

    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}

```

- 查找 [WebService(Namespace = "http://tempuri.org/")] 行，将 "http://tempuri.org/" 更改为 "http://tempuri.org/"。

运行 C# Web 服务

- 编译和运行服务。在 Web 浏览器中键入 <http://localhost/service.asmx>，其中 **localhost** 是 IIS Web 服务器的名称，**Service** 是服务的名称，在本例中为 `Service`。
- 输出为：

```

The following operations are supported. For a formal definition, please review the Service Description.
HelloWorld

```

- 单击 `HelloWorld` 链接以调用 `Service1` 的 `HelloWorld` 方法。输出为：

```

Click here for a complete list of operations.
HelloWorld
Test
To test the operation using the HTTP POST protocol, click the 'Invoke' button.

SOAP 1.1
...
SOAP 1.2
...
HTTP POST
...

```

- 单击“调用”按钮以调用 `Service1` 的 `HelloWorld` 方法。输出为：

```

<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://HowToDevelopWebServicesTest/">Hello World</string>

```

有关 Web 服务的更多信息，请参见：

- [生成 XML Web 服务客户端](#)
- [XML Web 服务描述](#)
- [如何：创建 XML Web services 方法](#)

- 演练:使用 Visual Basic 或 Visual C# 创建 XML Web services
- 演练:在 Visual Web Developer 中创建和使用 ASP.NET Web 服务

有关从 Java 到 C# 的自动转换的更多信息, 请参见 [Java Language Conversion Assistant 3.0 中的新增功能](#)。

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C#\(针对 Java 开发人员\)](#)

移动设备和数据(C# 与 Java)

有了 C# 和 .NET Compact Framework, 就可以使用桌面数据库编程所用的相同概念和类似 API 来访问和管理移动设备上的数据库数据。ADO.NET 在移动设备上提供一个面向 Windows CE 设备(包括 Pocket PC 和 Smartphone)的桌面 API 子集。有关更多信息, 请参见[数据库访问\(C# 与 Java\)](#)。

Java

在 Java 中, 可以使用 J2ME 和 JDBC 从移动设备访问数据库。有关更多信息, 请参见[数据库访问\(C# 与 Java\)](#)。J2ME 不会在所有设备上都表示为单一 API, 也没有单一的开发环境。另外, J2ME 必须在虚拟机上运行, 具体为 KVM 还是为 JVM 取决于配置。

C#

在 C# 中, 可以在桌面或移动设备上使用熟悉的连接、命令和数据表等概念来执行数据库读取操作。只需使用 `System.Data.SqlServerCe` 命名空间和类。例如, 可以执行下列操作:

- 将 `SqlCeConnection` 用于数据库连接。
- 将 `SqlCeCommand` 用于 SQL 命令对象。
- 将结果集对象(如 `DataTable`)用于数据表对象。

.NET Framework 提供了 `DataAdapter`, 使您可以轻松地将上述各类结合使用。可以使用 `SqlCeDataAdapter` 对象的连接属性来设置 `SqlCeConnection` 对象。

使用 `DataAdapter` 的 `SelectCommand` 属性或直接将其与连接对象一起传递给 `DataAdapter` 的构造函数, 以指定要执行的查询。

C#

```
da = new SqlCeDataAdapter("SELECT * FROM Users", cn);
```

`DataTable` 对象是使用 `DataAdapter` 对象的 `Fill` 方法创建的。`DataTable` 对象包含由查询返回的结果集数据。可以循环访问 `DataTable` 对象, 以便使用 `Rows` 集合访问数据行。

下面的示例演示如何访问移动设备上的 SQL Server CE (SQLCE) 数据库中的表行。

C#

```
namespace DataAccessCE
{
    using System.Data;
    using System.Data.SqlServerCe;

    class DataAccessCE
    {
        public static string connectionString = "";
        public static SqlCeConnection cn = null;
        public static SqlCeDataAdapter da = null;
        public static DataTable dt = new DataTable();

        static void Main()
        {
            connectionString = "Data Source=\\My Documents\\Database.sdf";
            cn = new SqlCeConnection(connectionString);

            da = new SqlCeDataAdapter("SELECT * FROM Users", cn);
            da.Fill(dt);

            foreach (DataRow dr in dt.Rows)
            {
                System.Console.WriteLine(dr[0]);
            }
        }
    }
}
```

有关更多信息，请参见：

- [ADO.NET 概述](#)
- [访问数据 \(Visual Studio\)](#)
- [ADO.NET 示例应用程序](#)
- [复制](#)
- [同步订阅](#)

有关从 Java 到 C# 的自动转换的更多信息，请参见 [Java Language Conversion Assistant 3.0 中的新增功能](#)。

编译代码

使用应用程序的 SQLCE 数据库前，需要将对 **System.Data.SqlClient** 的引用添加到项目中。在开发环境中，可在“项目”中选择“添加引用”，完成此操作。然后从“添加引用”对话框中选择 **System.Data.SqlClient** 组件。

注意

显示的对话框和菜单命令可能会与帮助中描述的不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

可靠编程

若要编译和运行代码，需要满足以下条件；否则，行 `da.Fill(dt)`；会失败并引发异常。

- 设备上安装了 SQL Server CE。
- 一个数据库表，其中包含用于在称为 `Database.sdf` 的 SQLCE 数据库中进行测试的数据。可以使用 SQL CE 工具在设备上生成此表，或从 SQL Server 桌面复制以生成此 `.sdf` 文件。可以将 `.sdf` 文件添加到项目，或将它手动复制到连接字符串中指定的目录。

请参见

参考

[智能设备](#)

[SqlConnection](#)

[SqlCommand](#)

概念

[C# 编程指南](#)

[ADO.NET DataSet](#)

其他资源

[C#\(针对 Java 开发人员\)](#)

[智能设备开发](#)

[在 ADO.NET 中使用 DataSet](#)

[ADO.NET for the Java Programmer \(ADO.NET\(针对 Java 程序员\)\)](#)

C# 应用程序类型(针对 Java 开发人员)

C# 应用程序类型包括 Windows 控制台应用程序、Windows 窗体应用程序、ASP.NET Web 应用程序、ASP.NET Web 服务应用程序、智能设备应用程序、ActiveX 应用程序以及安装和部署应用程序。

控制台应用程序

控制台应用程序使用标准命令行输入和输出(而不是使用窗体)进行输入和输出。控制台应用程序使用 `System.IO` 类处理输入和输出。您可以在方法之前加上类名(如 `System.IO.Console.WriteLine()`)，也可以在程序的起始处添加 `using` 语句。使用 Visual Studio 及其他开发环境(包括任何文本编辑器，如 Microsoft® 记事本)可以很容易地创建控制台应用程序。有关更多信息，请参见[Visual Studio 简介](#)、[创建控制台应用程序 \(Visual C#\)](#)、[Hello World -- 您的第一个程序 \(C# 编程指南\)](#)或[Main\(\) 和命令行参数 \(C# 编程指南\)](#)。

窗体应用程序

窗体应用程序采用的是用户熟悉的 Windows 图形用户界面，其中包含各种输入控件(如按钮和列表框)。窗体应用程序使用 `System.Windows.Forms` 命名空间中的类。使用 Visual Studio 及其他开发环境(包括任何文本编辑器，如 Microsoft® 记事本)可以很容易地创建窗体应用程序。有关创建 Windows 应用程序的更多信息，请参见[如何: 创建 Windows 应用程序项目](#)、[创建 ASP.NET Web 应用程序 \(Visual C#\)](#)或[创建 ASP.NET Web 应用程序 \(Visual C#\)](#)。

ASP.NET Web 应用程序

ASP.NET 应用程序是显示在 Web 浏览器中的 Web 应用程序，它不出现在控制台上或窗体应用程序中。ASP.NET 应用程序使用 `System.Web` 命名空间和诸如 `System.Web.UI` 这样的类来处理浏览器的输入和输出。您可以在方法之前加上类名(如 `using System.Web.UI.HtmlControls;`)，也可以在程序的起始处添加 `using` 语句。使用 Visual Studio 及其他开发环境(包括任何文本编辑器，如 Microsoft® 记事本)可以很容易地创建 ASP.NET 应用程序。有关创建 ASP.NET 应用程序的更多信息，请参见[Visual Web Developer](#)。有关使用 Visual Studio .NET 创建 ASP.NET 应用程序的更多信息，请参见[应用程序关系图上的 ASP.NET 应用程序概述](#)。有关 ASP.NET 的更多信息，请参见[.NET Framework 中的 ASP.NET Web 应用程序](#)。有关如何调试 ASP.NET 应用程序的更多信息，请参见[调试 ASP.NET Web 应用程序](#)和[调试准备: ASP.NET Web 应用程序](#)。

ASP.NET Web 服务应用程序

可以使用 URL、HTTP 和 XML 来访问 ASP.NET Web 服务，以便在任何平台上、以任何语言运行的程序均可访问 ASP.NET Web 服务。ASP.NET Web 服务应用程序可以显示在控制台上和窗体中，或者显示在 Web 浏览器或智能设备中。ASP.NET Web 服务应用程序使用 `System.Web` 和 `System.Web.Services` 命名空间和类。使用 Visual Studio 及其他开发环境(包括任何文本编辑器，如 Microsoft® 记事本)可以很容易地创建 ASP.NET 应用程序 Web 服务。有关创建 Web 服务应用程序的更多信息，请参见[访问和显示数据 \(Visual C#\)](#)和[如何: 创建 ASP.NET Web 服务项目](#)。有关在现有项目中添加 ASP.NET Web 服务的更多信息，请参见[如何: 使用托管代码向现有 Web 项目添加 XML Web services](#)。有关 ASP.NET Web 服务的更多信息，请参见[演练: 在 Visual Web Developer 中创建和使用 ASP.NET Web 服务](#)和[演练: 使用 Visual Basic 或 Visual C# 创建 XML Web services](#)。有关如何调试 ASP.NET Web 服务应用程序的更多信息，请参见[调试准备: XML Web services 项目](#)。

与 ASP.NET Web 服务相关的其他主题包括：

- [生成 XML Web 服务客户端](#)
- [XML Web 服务描述](#)
- [如何: 创建 XML Web services 方法](#)
- [演练: 使用 Visual Basic 或 Visual C# 创建 XML Web services](#)
- [演练: 在 Visual Web Developer 中创建和使用 ASP.NET Web 服务](#)
- [如何使用 Visual Studio .NET 2003 生成和测试 XML Web 服务](#)

智能设备应用程序

智能设备应用程序运行在移动设备(如 PDA 和 Smartphone)上。智能设备应用程序可以是控制台应用程序、Windows 窗体应用程序或 ASP.NET 和 Web 客户端，并显示在控制台上、窗体中或 Web 浏览器中。智能设备应用程序使用与桌面应用程序相同的命名空间和类。不过，智能设备应用程序使用 `Compact Framework`，而不使用 .NET Framework。有关在 Windows 移动设备上进行开发与面向桌面进行开发的详细对比，请参见[Developing Device vs. Desktop Applications \(开发设备与桌面应用程序\)](#)。有些版本的开发环境可能支持在移动设备上开发某些或所有类型的 C# 应用程序。有关创建 ASP.NET 应用程序的更多信息，请参见["How to create a Smart Device Application" \(如何创建智能设备应用程序\)](#)和["Smart Device Application Wizard" \(智能设备应用程序向导\)](#)。

与 ASP.NET Web 服务相关的其他主题包括：

- ["Use Visual Studio .NET 2003 to build and test a mobile Web application" \(使用 Visual Studio .NET 2003 生成和测试移动 Web 应用程序\)](#)
- [开发智能设备应用程序](#)
- ["Smart Device Programmability Features of Visual Studio .NET" \(Visual Studio .NET 的智能设备可编程性功能\)](#)
- [智能设备硬件注意事项](#)
- [ASP.NET 移动控件快速入门](#)
- [ASP.NET 移动控件示例](#)
- ["Smart device Walkthroughs" \(智能设备演练\)](#)
- [演练: 创建用于设备的 Windows 窗体应用程序](#)
- [演练: 调试设备项目中的 Windows 窗体](#)

ActiveX 控件

与 Java Bean 类似, ActiveX 控件也是一个组件, 并且与“OLE 对象”和组件对象模型 (COM) 对象等效。简而言之, ActiveX 控件就是一个支持 [IUnknown](#) 接口的 COM 对象。ActiveX 控件是一个开发在多种容器内重复使用的可编程软件组件的主要结构, 这些容器的范围从 Internet Explorer 到软件开发工具及最终用户生产工具。有关 ActiveX 控件的更多信息, 请参见:

- “[Introduction to ActiveX](#)”(ActiveX 简介)
- “[Packaging ActiveX Controls](#)”(将 ActiveX 控件打包)
- “[Using ActiveX Controls to Automate Your Web](#)”(使用 ActiveX 控件使 Web 自动执行操作)
- “[Designing Secure ActiveX Controls](#)”(设计安全的 ActiveX 控件)
- “[Using ActiveX Controls with Windows Forms in Visual Studio .NET](#)”(在 Visual Studio .NET 中将 ActiveX 控件用于 Windows 窗体)
- [调试 ActiveX 控件](#)

安装和部署应用程序

Visual Studio 为部署桌面、Web 及智能设备安装和部署项目提供模板。不同版本的开发环境可能支持在桌面、Web 及移动设备上安装和部署某些或所有类型的 C# 应用程序。有关更多信息, 请参见:

- [部署 .NET Framework 应用程序](#)
- [分发设备应用程序](#)
- [设备上的应用程序安装](#)
- “[Walkthrough: Generating Custom CAB files for Device projects](#)”(演练:为设备项目生成自定义 CAB 文件)
- “[Cabinet Packaging : Internet Explorer Code Download and the Java Package Manager](#)”(压缩打包:Internet Explorer 代码下载和 Java 程序包管理器)

相关主题

- [.NET Framework 网络操作基础](#)
- [Windows 窗体应用程序基础知识](#)
- [Java Language Conversion Assistant 3.0 中的新增功能](#)
- [.NET Framework 示例](#)

请参见

概念

[C# 编程指南](#)

其他资源

[迁移到 Visual C#](#)

[C# 代码示例\(针对 Java 开发人员\)](#)

[C# 编程语言\(针对 Java 开发人员\)](#)

[Visual C# 入门](#)

[使用 Visual C# IDE](#)

将 Java 应用程序转换为 Visual C#

Microsoft Visual Studio 2005 提供将在 Visual J++® 6.0 版中创建的项目或用 Java 语言编写的项目转换为 Visual C#® 项目的能力，以便您可以利用 .NET Framework。Java Language Conversion Assistant 可以从现有的 Visual J++ 6.0 项目或 Java 语言文件生成新的 Visual C# 项目。Java Language Conversion Assistant 向导使得转换现有文件的过程变得更加容易。

本节内容

[Java Language Conversion Assistant 3.0 中的新增功能](#)

此版本的 Java Language Conversion Assistant 包含以下功能：

[将 Visual J++ 或 Java 语言的项目转换为 Visual C#](#)

描述如何使用 Java Language Conversion Assistant。

[转换 Web 应用程序](#)

说明在使用 Java Language Conversion Assistant 转换 Web 应用程序时，如何改写这些应用程序。

[各种类型的 Java 语言应用程序的转换](#)

讨论要遵循的过程以及使用 Java Language Conversion Assistant 转换各种类型的应用程序时可能出现的问题。

[按包列出的 JLCA 诊断消息](#)

列出了 Java Language Conversion Assistant 生成的错误信息列表。

相关章节

[Java Language Conversion Assistant 向导](#)

描述如何使用该向导。

[JSP 自定义标记库的转换](#)

提供一列具有支持类的 Java 语言类，以模拟自定义标记的运行时行为。

[在打开已转换的 Servlet 类之前对这些类进行编译](#)

解释如何为已转换的 servlet 类打开 Web 窗体。

[疑难解答：匹配代码页](#)

说明在缺失代码页或代码页不匹配的情况下应如何操作。

[手动升级未转换的代码](#)

解释如何升级未能自动转换的代码。

[C# 参考](#)

提供 Visual C# 语言的概述。

Java Language Conversion Assistant 3.0 中的新增功能

此版本的 Java Language Conversion Assistant 包含以下功能：

- 支持 EJB 应用程序的转换
- 支持 CORBA 应用程序的转换
- 支持 RMI 应用程序的转换
- 支持 JMS 应用程序的转换
- 支持序列化应用程序的转换
- 支持使用 JNDI 的应用程序的转换
- 支持 JAAS 应用程序的转换
- 支持 JCE 应用程序的转换
- 支持 Java Swing 应用程序的转换
- 支持 JAXP 应用程序的转换
- 支持 TRAX 应用程序的转换
- 支持使用 JavaMail 的应用程序的转换
- 一个命令行属性开关

EJB

Enterprise JavaBeans 已转换为 [System.EnterpriseServices](#) 命名空间中的类，以支持 Enterprise JavaBeans 的所有类型：消息驱动 Beans、会话 Beans 和实体 Beans。

CORBA

通用请求代理体系结构 (CORBA) 类已转换为 [System.Runtime.Remoting](#) 命名空间中的类，以提供对分布式计算的支持。

RMI

远程方法调用 (RMI) 类已转换为 [System.Runtime.Remoting](#) 命名空间中的类，以提供对分布式应用程序和远程对象的支持。

JMS

Java 消息服务 (JMS) 类已转换为 [System.Messaging](#) 命名空间中的类，转换后的类使用 Windows 消息队列。

序列化应用程序

实现 [java.io.Serializable](#) 接口的类已转换为实现 [System.Runtime.Serialization.ISerializable](#) 接口。

使用 JNDI 的应用程序

Java 命名和目录接口 (JNDI) 类已转换为 [System.DirectoryServices](#) 命名空间中的类，以提供命名和目录服务。某些方法已转换为 [System.Runtime.Remoting](#) 方法，以支持 RMI 和 CORBA 的远程操作。

JAAS

Java 验证和授权服务 (JAAS) 类已转换为 [System.Security](#) 命名空间中的类，以提供所有安全与验证服务。

JCE

Java 加密扩展 (JCE) 类已转换为 [System.Security.Cryptography](#) 命名空间中的类，以提供消息的加密和解密，从而获得更高的安全性。

Java Swing

[javax.swing](#) 包中的类已转换为 [System.Windows.Forms](#) 命名空间中的类，以提供用户界面组件和控件。

JAXP

用于 XML 处理的 Java API 类已转换为 [System.Xml](#) 命名空间中的类。同时支持 SAX 和 DOM 模型。

TRAX

用于 XML 的转换 API 类已转换为 [System.Xml.Xsl](#) 命名空间中的类。

使用 JavaMail 的应用程序

JavaMail API 类已转换为 [System.Web.Mail](#) 命名空间中的类，以支持使用简单邮件传输协议 (SMTP) 构造和发送邮件。

属性开关

提供了一个命令行开关 (**/ProcessGetSetOff**)，您可以利用这个开关选择是将 **get/set/is** 方法保留为方法，还是将这些方法转换为属性。默认情况下，JLCA 将这些方法转换为属性。如果您愿意，可以切换命令行开关并将其中的大部分方法保留为方法。为了与其他转换兼容，下列模式始终转换为属性：

- ActiveX 方法转换为 .NET Framework 中的特定属性。
- CORBA 属性。
- .NET Framework 要求的访问器、转变器和状态检查方法。
- 自动生成的 .NET Framework 属性。

对于使用 **setProperty** 方法的 JSP 页面，如果其属性设置为通配符 (*)，则该页面不会正确转换，原因是用于模拟 Bean 属性设置的 SupportClass 代码需要的是一个属性，而不是 **get/set** 方法。

Taglib 处理程序类属性被转换为 **get/set** 方法，而在生成的 ASPX 页面中用于设置属性值的代码将无法执行，原因是自定义控件使用属性 (Property) 映射到自定义 HTML 标记中的 ASPX 属性 (Attribute)。

如果在一个线程中调用某一可视组件的访问器方法，而调用线程并不是拥有这一组件的线程，则该方法被转换为 **Invoke** 方法；转换后的方法在 **ProcessGetSetOff** 标志打开时会返回一个 **object** 类型。返回值必须强制转换为与原始方法的返回值相同的类型。

不支持的 Java 语言包

由于 CORBA 与 .NET Framework 远程处理在结构上存在差异，因此不支持以下 CORBA 包和类：

- **org.omg.CosNaming**
- **org.omg.CosNamingContextExtPackage**
- **org.omg.Dynamic**
- **org.omg.IOP**
- **org.omg.IOP.CodecFactoryPackage**
- **org.omg.IOP.CodecPackage**
- **org.omg.Messaging**
- **org.omg.PortableInterceptor**
- **org.omg.PortableInterceptor.ORBInitInfoPackage**

由于 Swing 与 Windows 窗体在结构上存在差异，因此不支持以下 Swing 包：

- **javax.swing.plaf**
- **javax.swing.plaf.basic**
- **javax.swing.plaf.metal**
- **javax.swing.plaf.multi**

由于 JLCA 包含用于实现第三方驱动程序的接口，因此不支持以下包：

- **javax.sql**

[请参见](#)

[参考](#)

[命令行开关](#)

[其他资源](#)

[将 Java 应用程序转换为 Visual C#](#)

[各种类型的 Java 语言应用程序的转换](#)

将 Visual J++ 或 Java 语言的项目转换为 Visual C#

使用 Java Language Conversion Assistant 可以转换 Visual J++ 6.0 项目和 Java 语言文件，以便使用 Visual C# 进行进一步的开发。Java Language Conversion Assistant 不对现有项目进行修改，而是基于原始项目创建新的 Visual C# 项目。新项目生成后，会在转换报告中显示转换过程中生成的任何错误、警告或问题。这些错误、警告和问题还会在已转换的代码的注释中注明，以帮助您手动转换项目中无法自动转换的部分。

注意

如果您对 Visual C# 并不熟悉，请在尝试进行转换之前，先花一些时间来熟悉该语言。有关更多信息，请参见 [C# 语言和 .NET Framework 介绍](#)。

注意

显示的对话框和菜单命令可能会与帮助中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

Visual J++ 项目

转换 Visual J++ 项目

1. 启动 Visual Studio。
2. 在“文件”菜单上，指向“打开”，然后单击“转换”。
3. 选择“Java Language Conversion Assistant”，然后单击“确定”。
4. 在“源文件”页中，选择“Visual J++ 6.0 项目”。
5. 在“选择项目”页中，单击“浏览”。
6. 浏览至正确的 .vjp 文件并选中它。

注意

如果 .vjp 文件的 CLASSPATH 指令指定 jar 或 .class 文件，将忽略这些文件。

7. 在“为新项目指定目录”页中，指定要创建的新项目的名称和目录。
8. 在“开始转换”页中，单击“下一步”。

Java 语言项目

转换 Java 语言项目

1. 启动 Visual Studio。
2. 在“文件”菜单上，指向“打开”，然后单击“转换”。
3. 选择“Java Language Conversion Assistant”，然后单击“确定”。
4. 在“源文件”页中，单击“包含项目文件的目录”。
5. 在“选择源目录”页中，单击“浏览”。
6. 浏览至正确的项目并选中它。

注意

您在所选目录中将看不到这些文件，但该目录中的所有 jav 和 java 文件都将转换。Java Language Conversion Assistant 将忽略大多数不相关的文件，但在开始转换之前应清理源文件夹。

7. 在第二个文本框中添加项目所需的其他任何文件。

8. 在“配置您的新项目”页中，指定以下内容：

- 要创建的项目的名称。
- 项目的输出类型。

9. 在“为新项目指定目录”页中，指定要创建的新项目的名称和目录。

10. 在“开始转换”页中，单击“下一步”。

11. 修复未能自动转换的代码。有关更多信息，请参见[手动升级未转换的代码](#)。

请参见

任务

[手动升级未转换的代码](#)

[Java Language Conversion Assistant 向导](#)

[疑难解答: 匹配代码页](#)

参考

[转换对话框](#)

[命令行开关](#)

概念

[Java Language Conversion Assistant](#)

Java Language Conversion Assistant 向导

Java Language Conversion Assistant 向导用于将 Visual J++ 和 Java 语言项目转换为 Visual C#。利用该向导，您可以创建一个新项目，然后将要从 Visual J++ 或 Java 语言转换为 Visual C# 的原始项目的文件复制进来。同时生成一个详细记录在转换过程中遇到的所有错误、警告或问题的报告。这些错误、警告和问题也会在新项目代码的注释中注明，您可以在任务列表中查看这些注释。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

访问 Java Language Conversion Assistant 向导

1. 在“文件”菜单上，指向“打开”，然后单击“转换”。
2. 单击“Java Language Conversion Assistant”，然后单击“确定”。

安全说明 您应该查看使用 Java Language Conversion Assistant 工具从 Java 转换的所有 C# 代码，确定其中是否有安全问题。任何不安全的 Java 代码都会被转换为不安全的 C# 代码。此外，该工具不会迁移某些 Java 类的代码，包括与安全性（例如身份验证）相关的某些类。在这些情况下，升级报告会说明哪些代码没有被迁移。因此您需要查看升级报告并减少任何安全问题，这一点非常重要。

请参见

任务

[将 Visual J++ 或 Java 语言的项目转换为 Visual C#](#)

[手动升级未转换的代码](#)

概念

[Java Language Conversion Assistant](#)

Java Language Conversion Assistant

Java Language Conversion Assistant 是用于将 Visual J++ 6.0 项目和 Java 语言文件转换为 Visual C# 的一种工具。通过将这些文件转换为 Visual C#, 您能够在利用现有代码库的同时, 还能利用 .NET Framework 所带来的好处。

新生成的 Visual C# 项目包含能够从现有 Visual J++ 或 Java 语言代码自动生成的所有新 Visual C# 代码。有关更多信息, 请参见[将 Visual J++ 或 Java 语言的项目转换为 Visual C#](#)。

您可以使用 Java Language Conversion Assistant 来转换 Visual J++ 或 Java 语言应用程序和小程序项目。它们以下列方式进行转换:

转换前	转换后
应用程序	Windows 窗体应用程序
小程序	Web 用户控件
JSP 页或 servlet	Web 应用程序

您可以在浏览器中承载转换后的 Web 用户控件, 就像在浏览器中承载小程序一样。寄宿的控件在 HTML 页中用 OBJECT 标记声明, 而不是用 APPLET 标记声明。使用 classid 属性标识控件, 方法是指定控件的路径和控件的完全限定名, 中间用英镑符号 (#) 分隔, 如下面的示例所示:

```
<OBJECT id="myControl" classid="http:ControlLibrary1.dll#ControlLibrary1.myControl" VIEWAS=EXT></OBJECT>
```

为了正确显示控件, 包含控件的 .dll 文件必须与显示控件的网页驻留在同一虚拟目录下, 或必须安装在全局程序集缓存中。

支持类

为了能够转换原始项目中 Visual C# 所没有的功能, Java Language Conversion Assistant 创建了用来重复原始功能的“支持类”(也称为“管理器”)。支持类有时与它们所模拟的类在结构上有很大的不同。尽管在转换后的项目中, 支持类会尽可能保留原应用程序的原始结构, 但它们的主要目的还是重复原始功能。

转换报告

在您的项目中可能会有一些未能自动转换的代码。在运行完 Java Language Conversion Assistant 向导后, 您可以查看转换报告, 该报告详细记录在转换过程中遇到的所有错误、警告和问题。在新生成项目的代码中, 未转换的源代码由标为 UPGRADE_TODO 的注释注明。您可以在任务列表中查看转换注释。每个转换注释都包含一个指向有关如何手动转换该代码的“帮助”主题的链接。有关更多信息, 请参见[手动升级未转换的代码](#)。

安全注意

为了安全起见, 您应当检查使用 Java Language Conversion Assistant 工具从 Java 转换的所有 C# 代码。任何不安全的 Java 代码都会被转换为不安全的 C# 代码。此外, 该工具不会迁移某些 Java 类的代码, 包括与安全性(例如身份验证)相关的某些类。在这些情况下, 升级报告会说明哪些代码没有被迁移。因此您需要查看升级报告并减少任何安全问题, 这一点非常重要。

请参见

任务

[将 Visual J++ 或 Java 语言的项目转换为 Visual C#](#)

[手动升级未转换的代码](#)

其他资源

[将 Java 应用程序转换为 Visual C#](#)

手动升级未转换的代码

Java Language Conversion Assistant 将 Visual J++ 项目或 Java 语言文件转换后，新的 Visual C# 项目可能包含未能自动转换的代码。在新项目代码中插入了注释，以帮助您将这些代码手动转换为 Visual C# 代码。

注释类型	说明
UPGRADE_TODO	未能自动转换的代码
UPGRADE_WARNING	可能有问题的代码
升级问题	错误的代码。
UPGRADE_NOTE	行为可能不同于源代码的代码

还可能有一些在编译应用程序前必须更正的编译器错误。每个注释都包含问题的简短说明和指向有关如何转换代码的“帮助”主题的链接。

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

手动升级未转换的代码

- 运行 Java Language Conversion Assistant 向导后，在 Visual Studio 中打开该项目。

请参见

任务

[将 Visual J++ 或 Java 语言的项目转换为 Visual C#](#)

概念

[Java Language Conversion Assistant](#)

疑难解答：匹配代码页

如果在转换之后发现代码中有奇怪的字符，则很可能是由于缺少用于字符编码的代码页或用于字符编码的代码页不匹配引起的。Java Language Conversion Assistant 以下列方式匹配编码：

- 在当前系统 ANSI 代码页中编码的文件在转换后保持此编码。
- 任何其他编码都被转换为 UTF-8 编码。

将以 Unicode 字节顺序标记开头的 Unicode 文件自动识别为 Unicode。如果文件不是以字节顺序标记开头，且未指定编码开关，则假定该文件在当前系统 ANSI 代码页中。

更改当前使用的系统 ANSI 代码页

1. 转到“控制面板”，然后双击“区域选项”(在 Windows 2000 中)或“区域和语言选项”(在 Windows XP 中)。
2. 单击“高级”，然后选择所需的代码页。

如果源文件使用的是非 ANSI 字符编码，或者您使用的是不是以字节顺序标记开头的 Unicode 源文件，则必须使用编码开关。

在编码开关中可以指定以下字符编码：

编码	开关
拉丁字母 2(中欧字符)	ISO-8859-2
拉丁字母 3	ISO-8859-3
拉丁字母 4(波罗的海沿岸各国字符)	ISO-8859-4
拉丁/西里尔字母	ISO-8859-5
拉丁/阿拉伯字母	ISO-8859-6
拉丁/希腊字母	ISO-8859-7
拉丁/希伯来字母	ISO-8859-8
拉丁字母 9	ISO-8859-15
日语	EUC-JP
朝鲜语	EUC-KR
简体中文	EUC-CN
中国国家标准字符	GB18030
无字节顺序标记的 UTF-8	UTF-8

若要使用特定的编码，必须在系统上安装对该编码的支持，例如：若要在 Windows 2000 Server 计算机上使用 GB18030 编码，必须下载并安装 GB18030 支持包。有关详细信息，请参阅您的系统文档。在使用编码开关时，项目中的所有文件都必须使用同一编码。

如果您使用的编码系统在 JLCA 中不受支持，则可能会丢失无法转换的字符。在某些情况下，还可能会丢失文件。

请参见

概念

[更改响应编码](#)

命令行开关

下表显示了 Java Language Conversion Assistant 3.0 中可用的命令行开关。

开关	功能
/?	输出消息。
/Out	指定目标目录。默认值为 \OutDir。
/Verbose	指定将输出所有内容。
/NoLogo	指定不显示版权标志。
/NoLog	指定不写入日志。
/LogFile	指定日志文件名。
/Encoding	指定输入文件的编码。
/ProjectName	指定目录转换的项目名称。
/ProjectType	指定项目类型。选项有 EXE、WinExe、Library 和 ASP.NET。默认值为 WinExe。
/VRoot	指定 Internet Information Server 的虚拟根目录。
/ContextPath	指定虚拟根目录要替换的上下文路径。
/Domain	指定要替换为 URL 的域。
/NoExtensibility	禁用可扩展性映射。
/JDK	指定要用于转换的 JDK。选项有 VJ 和 J2EE。
/ProcessGetSetOff	禁用属性转换。

属性开关

提供了一个命令行开关 (**/ProcessGetSetOff**)，您可以利用这个开关选择是将 **get/set/is** 方法保留为方法，还是将这些方法转换为属性。默认情况下，JLCA 将这些方法转换为属性。如果您愿意，可以切换命令行开关并将其中的大部分方法保留为方法。为了与其他转换兼容，下列模式始终转换为属性：

- ActiveX 方法转换为 .NET Framework 中的特定属性。
- CORBA 属性。
- .NET Framework 必需的访问器、增变器和状态检查方法。
- 自动生成的 .NET Framework 属性。

对于使用 **setProperty** 方法的 JSP 页面，如果其属性设置为通配符 (*)，则该页面不会正确转换，原因是用于模拟 Bean 属性设置的 SupportClass 代码需要的是一个属性，而不是 **get/set** 方法。

Taglib 处理程序类属性被转换为 **get/set** 方法，而在生成的 ASPX 页面中用于设置属性值的代码将无法执行，原因是自定义控件使用属性 (Property) 映射到自定义 HTML 标记中的 ASPX 属性 (Attribute)。

如果在其他不拥有可视化组件的线程中调用该组件的访问器方法，该方法将转换为 [System.Windows.Forms.Control.Invoke](#) 方法，后者在打开 **ProcessGetSetOff** 标志时会返回 **object** 类型。返回值必须强制转换为与原始方法的返回值相同的类型。

请参见

其他资源

[将 Java 应用程序转换为 Visual C#](#)

转换 Web 应用程序

将 Web 应用程序从 JSP 转换为 ASP.NET 时，需要其他信息。Java Language Conversion Assistant 会建立应用程序根目录，将 servlet 类移动到另一个目录，替换链接，并解析在 Web.xml 文件中定义的标记库说明符。可以为 Web 应用程序设置上下文路径和应用程序域以增强 URL 转换。

本节内容

[选择正确的 Web 应用程序转换类型](#)

说明转换类型以及如何为 Web 应用程序选择一个正确的转换类型。

[应用程序域和上下文路径设置](#)

说明 Web 应用程序转换的上下文路径和应用程序域设置。

[请求参数编码](#)

说明如何将 Java 语言请求参数编码转换为 ASP.NET。

[Web 应用程序中的目录结构差异](#)

说明 Java Language Conversion Assistant 如何修改 Web 应用程序的目录结构。

[链接替换](#)

说明 Java Language Conversion Assistant 如何替换转换后的 Web 应用程序中的链接。

[标记库说明符的解析](#)

说明如何转换 Web.xml 文件中包含的标记库说明符。

[更改响应编码](#)

说明如何更改使用非 UTF-8 编码的应用程序的响应编码。

[JSP 自定义标记库的转换](#)

说明 Java Language Conversion Assistant 如何处理 JSP 自定义标记库。

[在打开已转换的 Servlet 类之前对这些类进行编译](#)

解决 Servlet 类转换问题。

[Web 标记示例](#)

阐释各种 HTML 类的方法。

选择正确的 Web 应用程序转换类型

若要转换包含 JSP 页和/或 servlet 类的 Web 应用程序，必须在设置转换时选择正确的转换和输出类型。选择“Java 语言文件目录”，并将输出类型设置为 Web 应用程序。

[请参见](#)

[其他资源](#)

[转换 Web 应用程序](#)

应用程序域和上下文路径设置

转换 Web 应用程序时，您必须首先选择正确的输出类型：Web 应用程序。

当您输入了 Web 应用程序的虚拟根目录后，您就可以选择指定应用程序域，以增强应用程序中的链接转换方式。如果您没有指定应用程序域，所有链接（包括外部链接）都将转换为 ASP.NET 链接。

您还可以指定应用程序的上下文路径。

请参见

概念

[选择正确的 Web 应用程序转换类型](#)

其他资源

[转换 Web 应用程序](#)

请求参数编码

在 Java 语言中, servlet 容器使用 ISO 8859-1 编码格式对请求参数进行编码。如果您的 Web 应用程序处理不同的编码格式, 它通常包含以下某个示例中的代码。

示例

```
String text = request.getParameter("text");
text = new String( text.getBytes("ISO-8859-1"), charset )
```

下面的示例演示另一个版本:

```
String text = request.getParameter("text");
BufferedReader reader = new BufferedReader(new InputStreamReader(new StringBufferInputStream(text), charset));
text = reader.readLine();
```

在这两种情况下, *charset* 参数都是一个字符串, 表示您要解码的请求的编码。

在 .NET Framework 中, 此代码并不是必需的, 因为解码操作是由 ASP.NET 使用 Web.xml 文件中的参数集自动执行的, 如下面的示例所示。

示例

```
<globalization requestEncoding="euc-jp" responseEncoding="euc-jp"/>
```

转换 Web 应用程序时, 您可以移除实现此功能的已转换的 Java 语言代码, 因为已不再需要它们。

请参见

[其他资源](#)

[转换 Web 应用程序](#)

Web 应用程序中的目录结构差异

在转换 Web 应用程序时, Java Language Conversion Assistant 为新应用程序建立一个不同的目录结构。

所有 servlet 类都移动到 Web 应用程序根目录。在转换过程中创建的目录结构不一定对应于原始 Java 语言项目中 servlet 所在的目录。

示例

在下面的示例中, Web 应用程序根目录是 \website\AppName。servlet 类被移动到转换后的目录结构中的另一个目录。

原始目录结构

```
mt\src
mt\src\com\ais\servlets
mt\src\com\ais\servlets\Servlet1.java
mt\src\com\ais\servlets\Servlet2.java
mt\src\com\ais\beans
mt\src\com\ais\beans\CatalogBean.java
mt\src\com\ais\customtags
mt\src\com\ais\customtags\TableTag.java
mt\misc\pack1\myclasses
mt\misc\pack1\myclasses\DbHelper.java
mt\website
mt\website\AppName
mt\website\AppName\jsp\
mt\website\AppName\jsp\Search.jsp
mt\website\AppName\jsp\content\Catalog.jsp
mt\website\AppName\html\Index.html
mt\website\AppName\images\Logo.gif
mt\website\AppName\WEB-INF
mt\website\AppName\WEB-INF\Web.xml
mt\website\AppName\WEB-INF\MyTags.tld
```

转换后的等效目录

```
outdir\src
outdir\src\com\ais\beans
outdir\src\com\ais\beans\CatalogBean.cs
outdir\src\com\ais\customtags
outdir\src\com\ais\customtags\TableTag.cs
outdir\misc\pack1\myclasses
outdir\misc\pack1\myclasses\Dbhelper.cs
outdir\website
outdir\website\AppName
outdir\website\AppName\com\ais\servlets\Servlet1.aspx
outdir\website\AppName\com\ais\servlets\Servlet2.aspx
outdir\website\AppName\jsp\
outdir\website\AppName\jsp\Search.aspx
outdir\website\AppName\jsp\content\Catalog.aspx
outdir\website\AppName\html\Index.html
outdir\website\AppName\images\Logo.gif
outdir\website\AppName\WEB-INF
outdir\website\AppName\WEB-INF\Web.xml
outdir\website\AppName\WEB-INF\MyTags.tld
```

请参见

其他资源

[转换 Web 应用程序](#)

链接替换

Java Language Conversion Assistant 将替换所有链接，以用于转换后的目录结构。本节假定目录结构为 [Web 应用程序中的目录结构差异](#) 中定义的目录结构。

URL 和它们的更改方式

主要有以下三种类型的 URL：

- 带有域信息的绝对 URL

```
http://www.example.com/servlet/pack1/Servlet1
http://www.example.com/page.jsp
```

- 不带域信息的绝对 URL

```
/servlet/pack1/Servlet1
/directory/page2.jsp
```

- 相对 URL

```
pack1.Servlet1
page3.jsp
```

客户端用来访问 Web 应用程序的 URL 形式如下：

```
http://hoststring/ContextPath/resource
```

在本示例中，各元素的定义如下：

hoststring

映射到虚拟主机或 *hostname:portNumber* 的主机名。

ContextPath

访问应用程序所用的目录结构。

resource

在 Web.xml 配置文件中定义的对文件、JSP 页或 servlet 的引用。

Java Language Conversion Assistant 进行以下更改：

- 将上下文路径(如果存在的话)更改为包括以下两部分：在转换中定义的对应的虚拟根目录的名称，以及 Web 应用程序的根目录。
- 如果上下文路径不存在，则将 VROOT 插入到链接中。
- 如果文件扩展名是 .jsp，则将它更改为 .aspx。
- 使用 Web.xml 配置文件，通过查找 servlet 引用来转换他们。

示例

原始 URL

```
http://hoststring/ContextPath/dir/JSP1.jsp?param1=1
http://HostNameProvided/dir/JSP1.jsp?param1=1
```

等效的已转换 URL

```
http://hoststring/VRoot/dir/JSP1.aspx?param1=1
```

```
http://HostNameProvided/VRoot/dir/JSP1.aspx?param1=1
```

如果链接包含 servlet 标记和 servlet 的完整名称，则删去 servlet 标记并将完整名称中的圆点 (.) 更改为斜杠 (/)。如果指向 servlet 的链接是相对的，则将引用更改为包括以下两部分内容：VROOT 和转换后的 servlet 目录结构。

示例

原始 URL

```
http://hoststring/ContextPath/servlet/com.ais.servlets.Servlet1  
com.ais.servlets.Servlet1
```

等效的已转换 URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx  
/VRoot/com/ais/servlets/Servlet1.aspx
```

如果 servlet 标记的后面是 Web.xml 文件中的 servlet 名称，则将 servlet 的完整名称插入到链接中。

示例

原始 XML

```
<servlet>  
  <servlet-name>test</servlet-name>  
  <servlet-class>com.ais.servlets.Servlet1</servlet-class>  
</servlet>
```

原始 URL

```
http://hoststring/ContextPath/servlets/test
```

等效的已转换 URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx
```

如果 Web 应用程序名的后面是在 Web.xml 文件的 servlet 映射中定义的模式，则可以确定 servlet 的名称并将该名称插入到链接中。模式可以包括通配符 (*)。

示例

原始 XML

```
<servlet>  
  <servlet-name>test</servlet-name>  
  <servlet-class>com.ais.servlets.Servlet1</servlet-class>  
</servlet>  
<servlet-mapping>  
  <servlet-name>test</servlet-name>  
  <url-pattern>/*.asp</url-pattern>  
</servlet-mapping>
```

原始 URL

```
http://hoststring/ContextPath/abc.asp
```

等效的已转换 URL

```
http://hoststring/VRoot/com/ais/servlets/Servlet1.aspx
```

Java Language Conversion Assistant 不转换在运行时计算的链接。您必须手动更改代码以生成正确的 ASP.NET 链接。这适用于以下情况：

- 带有表达式 `<%= %>`、`jsp:forward`、`jsp:include` 或 `jsp:execute` 以及 HTML 标记 `<a href>` 和 `<form action>` 的链接，在这种链接中，URL 参数值在运行时进行计算。
- 参数表示 Web 资源路径的任何 `javax.servlet` 方法，例如，`Response.sendRedirect` 方法，在这种方法中，URL 参数值在运行

时进行计算。

示例

原始的 Java 语言代码

```
<jsp:include page="<% =value %>">
```

等效的 Visual C# 代码

```
<%-- //UPGRADE_TODO: Expected format of parameters of action jsp:include are different in t  
he equivalent in .NET Framework --%>  
<% Server.Execute(value)%>
```

请参见

概念

[Web 应用程序中的目录结构差异](#)

其他资源

[转换 Web 应用程序](#)

标记库说明符的解析

当 **taglib** 指令的 **uri** 属性引用在 Web.xml 文件中定义的 **taglib-uri** 时，标记库说明符 (TLD) 的位置可从该 Web.xml 文件中获得。

示例

原始 XML

```
<taglib>
    <taglib-uri>/MYTEST</taglib-uri>
    <taglib-location>WEB-INF/myTagLib.tld</taglib-location>
</taglib>
```

原始 JSP

```
<%taglib uri="/MYTEST" prefix="myTag"%>
```

要分析的等效 TLD

```
/WEB-INF/myTagLib.tld
```

请参见

其他资源

[转换 Web 应用程序](#)

更改响应编码

当您转换 Java 语言 Web 应用程序时，如果该应用程序包含输出编码不是 UTF-8 的 JSP 和/或 HTML 页，您可以通过修改 Web.config 文件，更改生成的 ASP.NET Web 应用程序的默认响应编码。查找 **Globalization** 元素，将 **responseEncoding** 属性更改为所需的编码。

[请参见](#)

[任务](#)

[疑难解答: 匹配代码页](#)

JSP 自定义标记库的转换

由于 JSP 自定义标记和 .NET Framework Web 用户控件之间存在差异，因此在转换过程中创建了一组模拟 JSP 自定义标记处理过程的支持类。这些类是转换后 JSP 自定义标记类处理程序的基础。为 **javax.servlet.jsp.tagext** 包中的以下类创建了支持类：

原始 Java 语言类	生成的支持类
BodyContent	WCBodyContent
Tag	WCBase
IterationTag	WCIterationBase
BodyTag	WCBodyBase
BodyTagSupport	WCBodyImpl
TagSupport	WCIterationImpl

请参见

其他资源

[转换 Web 应用程序](#)

在打开已转换的 **Servlet** 类之前对这些类进行编译

在转换 servlet 类时，您必须先解决所有转换问题并编译代码，然后才能为应用程序打开 Web 窗体页。在编译代码之前，不能在设计器视图中打开 Web 窗体页。如果您试图在编译之前打开它，则会出现一条错误消息。

[请参见](#)

[其他资源](#)

[转换 Web 应用程序](#)

Web 标记示例

下面的代码示例阐释 [System.Web.UI.HtmlTextWriter](#) 类的方法，并说明如何使用单个控件创建多个 HTML 元素：

示例

```
using System;
using System.Web;
using System.Web.UI;
using System.Collections.Specialized;
namespace CustomControls
{
    public class Rendered2 : Control, IPostBackDataHandler, IPostBackEventHandler
    {
        private String text1;
        private String text2;
        private String text = "Press button to see if you won.";
        private int number = 100;
        private int Sum
        {
            get
            {
                return Int32.Parse(text1) + Int32.Parse(text2);
            }
        }
        public int Number
        {
            get
            {
                return number;
            }
            set
            {
                number = value;
            }
        }
        public String Text {
            get
            {
                return text;
            }
            set
            {
                text = value;
            }
        }
        public event CheckEventHandler Check;
        protected virtual void OnCheck(CheckEventArgs ce)
        {
            if (Check != null)
            {
                Check(this,ce);
            }
        }
        public virtual bool LoadpostData(string postDataKey, NameValueCollection values)
        {
            text1 = values[UniqueID + "t1"];
            text2 = values[UniqueID+ "t2"];
            Page.RegisterRequiresRaiseEvent(this);
            return false;
        }
        public virtual void RaisePostDataChangedEvent() {
        }
        public void RaisePostBackEvent(string eventArgument)
        {
            OnCheck(new CheckEventArgs(Sum - Number));
        }
    }
}
```

```

        }
        protected override void Render(HtmlTextWriter writer)
        {
            writer.RenderBeginTag(HtmlTextWriterTag.H3);
            writer.Write("Enter a number:");
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Text");
            writer.AddAttribute(HtmlTextWriterAttribute.Name,this.UniqueID + "t1");
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"0");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.H3);
            writer.Write("Enter another number:");
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Text");
            writer.AddAttribute(HtmlTextWriterAttribute.Name,this.UniqueID + "t2");
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"0");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Br);
            writer.RenderEndTag();
            writer.AddAttribute(HtmlTextWriterAttribute.Type,"Submit");
            writer.AddAttribute( HtmlTextWriterAttribute.Name,this.UniqueID);
            writer.AddAttribute(HtmlTextWriterAttribute.Value,"Submit");
            writer.AddStyleAttribute(HtmlTextWriterStyle.Height,"25 px");
            writer.AddStyleAttribute(HtmlTextWriterStyle.Width,"100 px");
            writer.RenderBeginTag(HtmlTextWriterTag.Input);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Br);
            writer.RenderEndTag();
            writer.RenderBeginTag(HtmlTextWriterTag.Span);
            writer.Write(this.Text);
            writer.RenderEndTag();
        }
    }
}

//CheckEventArgs.cs.
//Contains the code for the custom event data class CheckEventArgs.
//Also defines the event handler for the Check event.
using System;
namespace CustomControls
{
    public class CheckEventArgs : EventArgs
    {
        private bool match = false;
        public CheckEventArgs (int difference)
        {
            if (difference == 0)
            {
                match = true;
            }
        }
        public bool Match
        {
            get
            {
                return match;
            }
        }
    }
    public delegate void CheckEventHandler(object sender, CheckEventArgs ce);
}

```

下面的代码示例演示如何使用 `System.Web.UI.WebControls.WebControl` 类的构造函数来创建 HTML `TextArea` 元素，并将其显示在 Web 窗体页上：

```
<font face="Courier New" size="2" color="#000080">
```

```
<%@ Page Language="C#" %>
<html>
<head>
<script runat="server">
void Button1_Click(Object sender, EventArgs e)
{
    WebControl wc = new WebControl(HtmlTextWriterTag.Textarea);
    PlaceHolder1.Controls.Add(wc);
}
</script>
</head>
<body>
<form runat="server">
<h3>WebControl Constructor Example</h3>
<p>
<asp:PlaceHolder id="PlaceHolder1" runat="Server" />
<br>
<asp:Button id="Button1" Text="Click to create a new TextArea" OnClick="Button1_Click" runat="Server" />
<p>
</form>
</body>
</html>
```

使用 [WriteBeginTag](#) 方法将指定 HTML 元素的任何制表位和开始标记写入到 **HtmlTextWriter** 输出流中。

此方法不写入 HTML 标记的结束字符 (>), 以便可以向元素添加 HTML 属性。可使用 [TagRightChar](#) 常数结束该标签。写入自结束 HTML 元素时, 可将 [WriteBeginTag](#) 方法与 [SelfClosingTagEnd](#) 常数一起使用。

此方法由自定义服务器控件使用, 这些控件不允许使用标记映射或属性映射, 并以同一方式为每个请求呈现 HTML 元素。

```
// Create a manually rendered tag.
writer.WriteBeginTag("img");
writer.WriteLineAttribute("alt", "AltValue");
writer.WriteLineAttribute("myattribute", "No \"encoding\" required", false);
writer.WriteLine(HtmlTextWriter.TagRightChar);
writer.WriteLineEndTag("img");
writer.WriteLine();
writer.Indent--;
writer.RenderEndTag();
```

您可以获取实例, 如下面的示例所示:

```
HtmlTextWriterTag myTag;
myTag = HtmlTextWriterTag.Area;
```

[请参见](#)

[参考](#)

[System.Web](#)

[HtmlTextWriter](#)

[WebControl](#)

各种类型的 Java 语言应用程序的转换

本节讨论在使用 Java Language Conversion Assistant 转换各种类型的应用程序时应遵循的过程和可能发生的问题。

本节内容

[JavaBeans 应用程序的转换](#)

解释 JavaBeans 的转换，包括必需的手动处理。

[EJB 应用程序的转换](#)

讨论在处理使用 Enterprise JavaBeans 技术的应用程序的转换时遇到的主要问题。

[CORBA 应用程序的转换](#)

解释 CORBA 应用程序的转换过程，包括 CORBA 和 .NET Framework Remoting 之间的结构差异。

[RMI 应用程序的转换](#)

说明远程方法调用 (RMI) 应用程序的转换。

[JMS 应用程序的转换](#)

讨论在转换 Java 消息服务 (JMS) 应用程序时遇到的问题。

[序列化应用程序的转换](#)

解释转换采用序列化的应用程序的过程。

[使用 JNDI 的应用程序的转换](#)

说明使用 Java 命名和目录接口 (JNDI) 的应用程序的转换问题。

[数据包的转换](#)

讨论数据库应用程序的转换。

[JAAS 应用程序的转换](#)

说明与 Java 身份验证和授权服务 (JAAS) 应用程序相关的转换问题。

[JCE 应用程序的转换](#)

说明加密应用程序的转换问题。

[Java Swing 应用程序的转换](#)

说明与 Java Swing 应用程序相关的转换问题。

[ActiveX 控件的转换](#)

说明与 Microsoft ActiveX 控件和 ActiveX Automation 对象相关的转换问题。

[Java.awt.swing 示例](#)

提供 Java Swing 应用程序转换的代码示例，包括 **IExtenderProvider** 接口的使用、用户定义事件以及 Windows Forms 控件的 **DataSource** 属性。

[如何: 设置远程处理的安全级别](#)

说明如何调整默认的安全级别以允许对象的序列化。

相关章节

[转换对话框](#)

说明可用来为项目选择转换工具的“转换”对话框。

[Web 标记示例](#)

提供使用 **System.Web.UI.HtmlTextWriter** 和 **System.Web.UI.WebControls** 类转换 Web 应用程序的代码示例。

[命令行开关](#)

说明 Java Language Conversion Assistant 中可用的命令行开关，特别是属性开关。

JavaBeans 应用程序的转换

在转换 JavaBeans 应用程序之前, 请确保它包含清单文件。否则, 它将被视为任何其他文件。清单文件的名称必须是 MANIFEST.MF 并且必须位于名为 META-INF 的文件夹中。如果需要, 可以重命名现有文件。

只有标准转换在清单文件中标识的 JavaBeans 才进行此类转换。例如:

```
Name: x\y class  
Java-Bean: True | False
```

在此示例中, x 表示包, y 表示 Bean 类组件。

Bean 的正确标识会影响它们转换到的类以及转换过程中关联的 **BeanInfo** 类中包含的信息的应用。例如, **JPanel** 对象通常转换为 [System.Windows.Forms.Panel](#) 对象。但是, 如果它在清单文件中被标识为 JavaBeans, 则它将转换为 [System.Windows.Forms.UserControl](#) 对象。

从 **java.awt.Panel** 或 **javax.swing.JPanel** 类继承的 Visual JavaBeans 被转换为从 **System.Windows.Forms.UserControl** 类继承。

转换过程中会应用关联的 **BeanInfo** 类中的信息, 但 .NET Framework 中没有这些类的直接等效项。

[请参见](#)

[参考](#)

[UserControl](#)

EJB 应用程序的转换

转换 Enterprise JavaBeans (EJB) 应用程序时，转换后的项目中只包括 EJB 相关的类和接口文件。如果应用程序在其他实用工具类上有依赖项，必须将它们手动添加到项目。默认情况下，所有实用工具类都包含在客户端项目中，并且可以在需要时手动排除。

.NET Framework 以不同的方式处理命名查找，会导致与命名上下文相关的编译错误。可以在 Visual C# 文件中注释掉相关代码。转换后必须手动设置组件安全性。

由于消息驱动的 bean 将转换为组件，所以必须验证所有将输出写入到控制台的代码。

部署组件时，必须使用强名称对 DLL 进行签名。必须生成密钥文件，并将其手动集成到程序集中。

部署后，检查已转换组件的事务设置。

如果要将组件发布到 Windows 2000 Server，必须移除所有服务组件的 **PrivateComponent** 标记。

安装客户端项目后，必须手动添加指向服务组件的引用。

事务设置

服务组件可以与一个事务属性相关联。在 Java 语言中，可以在方法级别应用事务属性。下表显示事务属性等效项：

EJB	.NET Framework
NotSupported	NotSupported
Supports	Supported
Required	Required
RequiresNew	RequiresNew
Mandatory	无 .NET Framework 等效项
Never	无 .NET Framework 等效项

转换期间，Java Language Conversion Assistant 试图根据 EJB 部署说明符中指定的事务属性以及每个属性与 EJB 方法相关联的次数，确定组件最有代表性的 .NET Framework 事务属性。如果为某个方法指定的 EJB 事务属性与为该组件选择的 .NET Framework 事务属性不等效，则将在 Visual C# 代码中生成警告。查找这些警告消息。

安全设置

必须以编程方式或通过组件服务资源管理器设置下列 .NET Framework 安全设置：

- 授权
- 安全级别
- 身份验证级别
- 模拟级别

必须启用“授权”以执行安全检查。设置 [System.EnterpriseServices.ApplicationAccessControlAttribute](#) 类属性以启用安全检查。这些属性用于配置其余设置（安全级别、身份验证级别和模拟级别）。

可以在处理级别或同时在处理级别和组件级别执行“安全检查”。[System.EnterpriseServices.AccessChecksLevelOption](#) 枚举定义下列选项：

- [Application](#)
- [ApplicationComponent](#)

只执行处理级别的安全检查时，将忽略和禁用接口级别、组件级别和方法级别的基于角色的安全设置。将已转换应用程序的安全级别设置为 **ApplicationComponent**。

“身份验证”级别控制对 .NET Framework 应用程序用户进行身份验证的方式。使用 [System.EnterpriseServices.AuthenticationOption](#) 枚举中的下列值之一：

- [System.EnterpriseServices.AuthenticationOption.None](#)
- [Connect](#)
- [Call](#)
- [Packet](#)
- [Integrity](#)
- [Privacy](#)

默认身份验证级别为 **Packet**。在此级别，将对来自用户的每个数据包进行身份验证。将已转换应用程序的身份验证级别设置为 **Packet**。

“模拟”是指当从原始应用程序访问另一个应用程序或受保护的资源时，传播客户端安全身份的方式。这样，服务器可以在某种程度上模拟客户端。[System.EnterpriseServices.ImpersonationLevelOption](#) 枚举提供下列选项：

- [Anonymous](#)
- [Default](#)
- [Delegate](#)
- [Identify](#)
- [Impersonate](#)

.NET Framework 应用程序的默认模拟级别为 **Impersonate**。此级别允许服务器担任客户端角色。不过，服务器不能代表该客户端访问其他计算机上的对象或资源。将已转换应用程序的级别设置 **Impersonate**。

问题

JLCA 将 EJB 项目转换为 COM 应用程序。如果 EJB 源文件的文件夹同时包含 JAR 文件和部署说明符 Ejb-jar.xml 文件，则因为也会对 JAR 文件中包含的部署说明符进行分析，将在已转换项目中将生成两个应用程序。

转换 EJB 时，会尝试将 .NET 事务属性分配给它的服务组件。对于一个 EJB，如果将不同的事务属性全局分配给本地接口和远程接口，则不会分配 .NET 事务属性。同样，如果一个接口被全局分配了事务属性，其他接口仅有特定方法分配，也不会分配 .NET 事务属性。仅在没有全局事务属性分配时，才使用特定方法事务属性分配确定 .NET 事务属性。因为 .NET 服务组件处理事务属性的方式与 Java 语言不同，所以需要检查设置，并进行必要的调整。

转换期间将添加默认的安全检查，而这种检查在项目中可能是不必要的。如果 Java 语言源代码将安全角色设置为特定方法，或排除某些方法，请在转换后检查设置。

请参见

参考

[System.EnterpriseServices](#)

CORBA 应用程序的转换

在转换 CORBA 应用程序之前, 请确保该应用程序具有 OMG IDL 文件。否则, Java 文件将被视为任意其他类。

转换

确保源文件的主干或存根 (stub) 代码中不包含任何必需的代码。因为这些代码将在转换过程中丢失。

指定为目标的目录必须包含在 CLASSPATH 中。

默认情况下, CORBA 应用程序转换为通过 HTTP 的 .NET Framework 远程处理。这意味着转换后的应用程序不再使用 CORBA 客户端和服务器。

.NET Framework 远程处理使用直接客户端/服务器连接, 没有任何干扰命名服务或中间层。因此, 不存在任何对象引用中间装置 (ORB)。

创建 CORBA 命名服务所需要的大部分代码在 .NET Framework 远程处理中不再需要。这部分代码包括通常打包为哈希表或属性数组的类和集合。

注释掉所有对下列包中的类的引用:

- **org.omg.CosNaming**
- **org.omg.CosNamingContextPackage**

CORBA 应用程序必须先获得命名上下文。Java Language Conversion Assistant 将此过程视为远程查找, 而在 .NET Framework 远程处理中无需这样做。下列代码行可被注释掉:

```
NamingContext ncRef = (org.omg.CosNaming.NamingContext) Activator.GetObject(typeof(org.omg.CosNaming.NamingContext), "http://<Host>:<Port>/<Name>");
```

安全问题

在 .NET Framework 中, 分布式通信(远程)的默认安全级别为 **low**。这会影响向远程方法传递用户定义的对象类型。有关更多信息, 请参见[如何:设置远程处理的安全级别](#)。

请参见

参考

[System.Runtime.Remoting](#)

概念

[使用 JNDI 的应用程序的转换](#)

RMI 应用程序的转换

在 Java 语言中，远程方法调用 (RMI) 包含三部分：服务器、客户端以及对象注册表。

当编译服务器端应用程序时，会生成两个文件：服务器端的“主干”文件和客户端的“存根”文件。服务器端的主干文件和客户端的存根文件均在内部使用，用于管理方法调用和数据传输。

将 RMI 客户端和服务器转换为 .NET Framework 不会产生 Visual C# RMI 客户端和服务器。.NET Framework Remoting 默认情况下用于 HTTP。由于方法调用和数据传输都是在内部处理的，因此不会生成可与主干和存根相比的额外文件。

RMI 客户端通过发送字符串 URL 来请求远程对象。此字符串具有以下形式：`rmi://host:port/name`，其中 `port` 是服务器用来注册远程对象的端口。（默认为 1099。）

在 .NET Framework 中，该字符串 URL 标识要使用的通道协议，其形式为：`:protocol://host:port/name`。`port` 是服务器为侦听客户端请求而注册的端口。

在 Java 语言中，如果一个对象符合以下两个条件，则将其标识为远程的：第一，该对象扩展了 `java.rmi.server.UnicastRemoteObject` 类；第二，该对象的所有方法引起 `RemoteException` 错误。几乎所有从 `RemoteException` 继承的异常都转换为 `System.Runtime.Remoting.RemotingException` 错误。如果您的应用程序以不同的方式处理特定的 `RemotingException` 子类，那么，您必须对这些子类进行手动调整。

在 .NET Framework 中，如果对象是从 `System.MarshalByRefObject` 类派生的，则可以将其标识为远程对象。

每个 RMI 接口在转换过程中都被放置在单独的 Visual Studio 项目中。必须更新项目引用，以包括远程接口程序集。如果客户端和服务器的入口点都包括在转换后的源树中，则必须将输出代码放在单独的 Visual Studio 项目中。客户端项目和服务器项目都需要引用共享的 DLL 程序集。

安全问题

在 .NET Framework 中，分布式通信（远程）的默认安全级别为 `low`。这会影响向远程方法传递用户定义的对象类型。有关更多信息，请参见[如何：设置远程处理的安全级别](#)。

请参见

参考

[RemotingException](#)

[MarshalByRefObject](#)

概念

[使用 JNDI 的应用程序的转换](#)

JMS 应用程序的转换

Java 消息服务 (JMS) 转换为使用 Windows 消息队列。因此，创建连接、连接工厂和会话的代码在转换过程中被注释掉。您必须移除代码中其他位置的相关变量。在 Visual C# 代码中忽略与属性和 JNDI 相关的代码。

发行者-订阅者模式的转换与点对点模式完全相同。

不支持以下 JMS 元素：

- 主题
- 持久订阅者
- 消息选择器
- 异常侦听器

必须选择以下机制之一，才能在 .NET Framework 中使用多目标消息：

- 通讯组列表
- 多元素格式名
- 多路广播地址

这些机制中的每一个都使用字符串值引用目标队列，并在 [System.Messaging.MessageQueue.Path](#) 属性中设置。

您必须手动更改引用队列或主题的字符串以使用 Windows 消息队列。**TemporaryQueue** 对象转换为物理队列（必须显式删除）。

.NET Framework 中处理事务队列的方式有所不同。您必须使用 [System.Messaging.MessageQueueTransaction](#) 对象设置事务性队列并重写事务管理。

.NET Framework 中的消息确认管理是自动进行的。如果您的代码中有手动消息确认，必须将其移除。

如果连接不可用，Windows 消息队列不会尝试再次连接。

XA 对象不转换。在 .NET Framework 中必须重写分布式事务。

转换的应用程序不能与 JMS 兼容 MOM 交互。

请参见

参考

[System.Messaging](#)
[MessageQueueTransaction](#)

序列化应用程序的转换

在 Java 语言中，通过此事实来识别可序列化的对象：它们直接或者通过继承来实现 `java.io.Serializable` 接口。.NET Framework 以类似的方法使用 `System.Runtime.Serialization.ISerializable` 接口，但也可以使用 `SerializableAttribute` 属性。因此，不可能在运行时以编程方式确定对象是否可以序列化。请小心处理 Java 语言代码中接收或返回 `Serializable` 接口的任何方法。它与 .NET Framework 中的等效项不能互换，它们不能相互读取或写入。

在 Java 语言中，可序列化的类通常实现 `readObject` 和 `writeObject` 方法。`writeObject` 方法转换为重载的构造函数，而 `readObject` 转换为 `GetObjectData` 方法。大多数情况下，转换的代码都可以直接编译和运行，不需要做任何更多的修改。不过，它使用自定义封送处理代码和取消封送处理代码，因此确实是类定义的一项主要更改。

请参见

参考

[ISerializable](#)

使用 JNDI 的应用程序的转换

相同的服务提供程序必须在两种环境中都可以使用，已转换的应用程序的功能才能保留。只有 LDAP/ActiveDirectory 在 Java 语言和 .NET Framework 中都受支持。

通常支持命名和目录服务的 JNDI 方法转换为 [System.DirectoryServices](#) 命名空间中的方法。通常支持 RMI 和 CORBA 的方法转换为 [System.Runtime.Remoting](#) 命名空间。由于某些方法可用于双重目的，因此一些 [System.Runtime.Remoting](#) 方法可能需要在已转换代码中更改为 [System.DirectoryServices](#) 方法。

在 Java 语言中，可以使用接受定义环境的哈希表的构造函数来实例化上下文。[System.DirectoryServices.DirectoryEntry](#) 类没有这样的构造函数。

标识提供程序的目录项路径区分大小写，必须更改，例如从 `ldap` 改为 `LDAP`。

在 Java 语言中，搜索参数以独立于上下文的方式被定义，然后传递给上下文搜索方法。在 .NET Framework 中，通过创建 [System.DirectoryServices.DirectorySearcher](#) 对象，搜索参数与要搜索的目录入口一起定义。必须修改代码以符合这种差异。

不使用 [DirContext.bind](#) 和 [DirContext.rebind](#) 方法，而是使用 [System.DirectoryServices.DirectoryEntry](#) 类及其关联的类在目录中添加或替换对象。

不使用 [Context.createSubcontext](#) 方法，而是使用

[System.DirectoryServices.DirectoryEntries.Add\(System.String,System.String\)](#) 方法，并设置新入口的属性。

在 [javax.rmi.naming](#) 包中具有等效项的所有 [Context](#) 方法均转换为 [System.Runtime.Remoting.RemotingServices](#) 方法。这些方法包括 [Context.bind](#)、[Context.lookup](#)、[Context.rebind](#) 和 [Context.unbind](#)。

不支持 [javax.naming.ldap](#) 包。

请参见

概念

[CORBA 应用程序的转换](#)

[RMI 应用程序的转换](#)

数据包的转换

使用 **java.sql** 和 **javax.sql** 包的应用程序被转换为 **System.Data.OleDb** 命名空间。如果您使用不同的数据库系统，则可以转换对系统的客户端的这些引用，例如 **System.Data.SqlClient** 或 **System.Data.ODBC**。

.NET Framework 数据提供程序自动进行池连接。可以为每个数据提供程序（例如 ODBC、OleDb 和 SQL Server）配置连接池。

您必须将所有连接字符串都转换为 .NET Framework 格式。以下示例显示如何转换不同类型的数据库连接。

示例

下面的示例显示如何使用 JDBC-ODBC 桥：

原始的 Visual J++ 代码

```
String dbUrl = "jdbc:odbc:BiblioODBC";
Connection c = DriverManager.getConnection(dbUrl);
```

等效的 Visual C# 代码

```
String dbUrl = "Provider = MSDASQL; DSN = BiblioODBC";
OdbcConnection c = "DriverManager.getConnection(dbUrl);
```

下面的示例显示如何使用来自 Microsoft SQL Server 的 URL：

原始的 Visual J++ 代码

```
String dbUrl = "jdbc:microsoft:sqlserver://MySqlServer:1433; DatabaseName=pubs";
Connection c = "DriverManager.getConnection(dbUrl, username, pwd);
```

等效的 Visual C# 代码

```
String dbUrl = "Provider = SQLOLEDB; DataSource = MySqlServer; Initial Catalog = pubs";
SQLConnection c = DriverManager.getConnection (dbUrl + "; User ID = " + username + "; PWD + " + pwd);
```

原始的 Oracle 连接字符串

下面的示例显示如何更改 Oracle 数据库连接字符串：

```
String jdbcUrl = "jdbc:oracle:thin:@localhost:1521:ORCL";
conn = DriverManager.getConnection(jdbcUrl, user, password);
```

等效的 Visual C# 代码

```
String jdbcUrl = "Provider = MSDAORA; Data Source = localhost:1521;";
OracleConnection conn = DriverManager.getConnection(jdbcUrl + "User ID = " user + "; Password = " + password);
```

可更新、可滚动的结果集不能可靠地转换。虽然提供了一个支持类，但它要求进行某些手动调整。如果您的应用程序十分依赖于此类结果集，请使用可以自动转换的 **System.Data.DataView** 或 **System.Data.DataTable** 类，而不使用 **System.Data.OleDb.OleDbDataReader** 类。

Clob 和 **Blob** 数据类型被分别转换为字符数组和字节数组，它们具有不同的行为并可能占用更多内存。转换后的代码按顺序获取数据并创建数组。您可能需要将数据流写入某个磁盘文件。您还可以使用 **System.Data.OracleClient.OracleLob** 类，它用于在 Oracle 服务器上表示 **Large Object Binary** 数据类型。在 .NET Framework 中不支持某些新的数据类型，例如 **STRUCT**、**ARRAY** 或用户定义的类型。托管的 **System.Data.OracleClient.OracleBFile** 类适用于 Oracle **BFile** 数据类型。

数据访问类型特定于数据库类型。因为 OleDb 驱动程序可用于大多数数据库，所以所有 JDBC 代码都被转换为 OleDb 类型。在某些实例中，要获得更好的性能或支持，您最好在转换后进行这一更改。在 .NET Framework 中可用的其他数据适配器包括 ODBC、SQL 和 Oracle。

DataSources 不被映射；不过，在某些情况下，**OleDbConnection** 或 **System.Data** 命名空间下的任何其他客户端连接类可用于表示某些功能。可以使用 **System.Diagnostics.EventLog** 类实现 **javax.sql.DataSource** 接口的日志记录功能。

javax.sql 包不能可靠地转换。该包中的接口必须由第三方驱动程序实现。

Driver 接口和 **DriverManager** 类已被否决，不再需要。它们的某些方法可被转换，但它们是不必要的。

请参见

参考

[System.Data.OleDb](#)

[System.Data.SqlClient](#)

[System.Data.Odbc](#)

[DataAdapter](#)

[DataTable](#)

[OleDbDataReader](#)

概念

[了解连接池](#)

[使用连接池](#)

JAAS 应用程序的转换

.NET Framework 使用具有主体类、标识类和权限类的基于角色的安全性来处理安全性。您可以选择 .NET Framework 中某个内置的安全模块，而不用自己生成模块。当您将 Java 身份验证和授权服务 (JAAS) 应用程序转换为 .NET Framework 时，必须考虑两种安全方法之间的差别。

所有 JAAS 配置文件都必须重命名为 JAAS.config 后才能由 Java Language Conversion Assistant 处理。这些配置文件转换为 App.config 文件，支持类方法可使用这些文件来获取身份验证模块并将这些模块注册到身份验证管理器中。

LoginContext 类转换为静态的 **System.Security.AuthenticationModule** 类，后者具有不同的行为。

LoginModule 类转换为 **IAuthenticationModule** 接口。在 Java 语言中，**LoginContext** 对象注册一个 **LoginModule** 对象，后者使用回调处理程序从用户和登录模块中请求输入，以便对用户进行身份验证。在 .NET Framework 中，身份验证模块注册到身份验证管理器中，后者循环通过已注册的身份验证模块以返回授权信息。

使用泛型 **System.Security.SecurityException** 代替不同的 JAAS 异常。

Subject 类转换为 **System.Security.Principal.GenericPrincipal** 类，后者具有不同的行为。还有一个用于 Windows 身份验证的 **System.Security.Principal.WindowsPrincipal** 类。

请参见

参考

[System.Security](#)

[System.Security.Principal](#)

[GenericPrincipal](#)

[WindowsPrincipal](#)

JCE 应用程序的转换

在 Java 语言中，泛型类或接口被用作所有加密类的基类。在 .NET Framework 中，[System.Security.Cryptography.AsymmetricAlgorithm](#) 类是所有公钥算法的基类。在转换过程中，**CryptoSupport** 支持类作为 [System.Security.Cryptography.SymmetricAlgorithm](#) 类的包装生成。您必须手动重新实现不对称算法。

在 Java 语言中，所有加密算法类型都由带有一个指示加密类型的字符串的 **Cipher.getInstance** 方法生成。每种加密类型在 .NET Framework 中使用一个单独的类。提供了一个相对有限的加密类集。

请参见

参考

[System.Security.Cryptography](#)

[SymmetricAlgorithm](#)

[AsymmetricAlgorithm](#)

Java Swing 应用程序的转换

Swing 应用程序转换为 [System.Windows.Forms](#) 命名空间。因此，以下 Swing 元素在 Visual C# 中没有直接的等效项，不能转换。

- 不支持 **javax.swing.plaf** 包。.NET Framework 中没有封装 Swing 可插入外观的类或引用它的方法的等效项。
- 不支持呈现器和编辑器。
- 不支持 Java AWT 布局。这些虽然不是 Swing 特有的，但它们会影响大多数 Swing 应用程序。

对以下元素的提供有限的支持，或者需要在 Java Language Conversion Assistant 转换后进行重要的手动改编：

- 仅部分支持模型。
- 诸如 **JTree** 和 **JTable** 这样的类由于对模型的依赖性，不能直接转换为它们在 .NET Framework 中的等效项。实现了支持类以帮助匹配这些类，但并不是所有项都可以映射。
- ContentPane** 类和其他中间级容器不能直接转换。例如，**JFrame** 对象在它的内容窗格中包含控件，而 .NET Framework [System.Windows.Forms.Form](#) 类在自身中包含控件，没有任何中间对象。这会导致继承问题，转换需要大量的用户输入。
- 行为由常数值指定的类不能完全自动转换。例如，**FileDialog** 类有一个指定它是打开还是保存文件的常数。在 Visual C# 中，有两个类分别处理此问题。
- 在 .NET Framework 中，事件处理建立在另一个模型之上，因此所有的 Swing 事件在转换后都需要手动改编。在 [Java Swing 示例](#) 主题中包含一个事件处理转换的示例。

请参见

参考

[Java Swing 错误信息](#)

[System.Windows.Forms](#)

概念

[Java Swing 示例](#)

ActiveX 控件的转换

com.ms.wfc.ax 包包含 Microsoft ActiveX 控件和 ActiveX 自动化对象的类和接口。

在 .NET Framework 中, Windows 窗体只能承载 Windows 窗体控件。因此必须将 ActiveX 控件包装在从 [System.Windows.Forms.AxHost](#) 类派生的包装类中, 以使其显得类似于 Windows 窗体控件。

可以通过下列任一种方式转换 ActiveX 组件:

- Visual Studio 自动将类型库中的 COM 类型转换为程序集中的元数据。
- 导入类型库提供命令行开关, 以调整元数据并生成交互操作程序集和命名空间。
- 自定义包装是工作量最大的选项, 但可以根据应用程序的需要对其进行定制。

[请参见](#)

[参考](#)

[AxHost](#)

Javax.swing 示例

下面的示例显示如何使用扩展程序提供控件属性(使用 [System.ComponentModel.IExtenderProvider](#) 接口)：

```

using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Globalization;
namespace Extenders
{
    [ProvideProperty("MyString", typeof(Control))]
    public class UserControl1 : System.Windows.Forms.UserControl, IExtenderProvider
    {
        private System.ComponentModel.Container components = null;
        public UserControl1()
        {
            InitializeComponent();
        }
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }
        #region Component Designer generated code
        private void InitializeComponent()
        {
            this.Name = "UserControl1";
            this.Size = new System.Drawing.Size(170, 160);
        }
        #endregion
        [Browsable(true)]
        public String GetMyString(Control control)
        {
            return "This is my string.";
        }
        public void SetMyString(Control container, String value)
        {
        }
        public bool CanExtend(Object control)
        {
            return true;
        }
    }
}

```

下面的示例显示如何实现处理和引发用户定义的事件：

```

namespace EventSample
{
    using System;
    using System.ComponentModel;
    //Class that contains the data for
    //the alarm event. Derives from System.EventArgs.
    public class AlarmEventArgs : EventArgs
    {

```

```

private readonly bool snoozePressed;
private readonly int nrings;
public AlarmEventArgs(bool snoozePressed, int nrings)
{
    this.snoozePressed = snoozePressed;
    this.nrings = nrings;
}
//The NumRings property returns the number of rings
//that the alarm clock has sounded when the alarm event
//is generated.
public int NumRings
{
    get { return nrings; }
}
}
//The SnoozePressed property indicates whether the snooze
//button is pressed on the alarm when the alarm event is generated.
public bool SnoozePressed
{
    get {return snoozePressed;}
}
//The AlarmText property that contains the wake-up message.
public string AlarmText
{
    get
    {
        if (snoozePressed)
        {
            return ("Wake up! Snooze time is over.");
        }
        else
        {
            return ("Wake Up!");
        }
    }
}
}
//Delegate declaration.
public delegate void AlarmEventHandler(object sender, AlarmEventArgs e);
//The Alarm class that raises the alarm event.
public class AlarmClock
{
    private bool snoozePressed = false;
    private int nrings = 0;
    private bool stop = false;
    //The Stop property indicates whether the
    //alarm should be turned off.
    public bool Stop
    {
        get {return stop;}
        set {stop = value;}
    }
    //The SnoozePressed property indicates whether the snooze
    //button is pressed on the alarm when the alarm event is
    //generated.
    public bool SnoozePressed
    {
        get {return snoozePressed;}
        set {snoozePressed = value;}
    }
    //The event member that is of type AlarmEventHandler.
    public event AlarmEventHandler Alarm;
    //The protected OnAlarm method raises the event by invoking
    //the delegates. The sender is always this, the current instance
    //of the class.
    protected virtual void OnAlarm(AlarmEventArgs e)
    {
        if (Alarm != null)

```

```

    {
        //Invokes the delegates.
        Alarm(this, e);
    }
}
//This alarm clock does not have
//a user interface.
//To simulate the alarm mechanism, it has a loop
//that raises the alarm event at every iteration
//with a time delay of 300 milliseconds,
//if snooze is not pressed. If snooze is pressed,
//the time delay is 1000 milliseconds.
public void Start()
{
    for (;;)
    {
        nrings++;
        if (stop)
        {
            break;
        }
        else if (snoozePressed)
        {
            System.Threading.Thread.Sleep(1000);
            {
                AlarmEventArgs e = new AlarmEventArgs(snoozePressed,
                nrings);
                OnAlarm(e);
            }
        }
        else
        {
            System.Threading.Thread.Sleep(300);
            AlarmEventArgs e = new AlarmEventArgs(snoozePressed, nrings);
            OnAlarm(e);
        }
    }
}
//The WakeMeUp class that has a method AlarmRang that handles the
//alarm event.
public class WakeMeUp
{
    public void AlarmRang(object sender, AlarmEventArgs e)
    {
        Console.WriteLine(e.AlarmText +"\n");
        if (!(e.SnoozePressed))
        {
            if (e.NumRings % 10 == 0)
            {
                Console.WriteLine("Let alarm ring? Enter Y");
                Console.WriteLine(" Press Snooze? Enter N");
                Console.WriteLine(" Stop Alarm? Enter Q");
                String input = Console.ReadLine();
                if (input.Equals("Y") ||input.Equals("y")) return;
                else if (input.Equals("N") || input.Equals("n"))
                {
                    ((AlarmClock)sender).SnoozePressed = true;
                    return;
                }
                else
                {
                    ((AlarmClock)sender).Stop = true;
                    return;
                }
            }
        }
    }
}

```

```

        {
            Console.WriteLine(" Let alarm ring? Enter Y");
            Console.WriteLine(" Stop alarm? Enter Q");
            String input = Console.ReadLine();
            if (input.Equals("Y") || input.Equals("y")) return;
            else
            {
                ((AlarmClock)sender).Stop = true;
                return;
            }
        }
    }
//The driver class that hooks up the event handling method of
//WakeMeUp to the alarm event of an Alarm object using a delegate.
//In a forms-based application, the driver class is the
//form.
public class AlarmDriver
{
    public static void Main (string[] args)
    {
        //Instantiates the event receiver.
        WakeMeUp w= new WakeMeUp();
        //Instantiates the event source.
        AlarmClock clock = new AlarmClock();
        //Connects the AlarmRang method to the Alarm event.
        clock.Alarm += new AlarmEventHandler(w.AlarmRang);
        clock.Start();
    }
}
}

```

下面的示例显示如何使用控件的 `DataSource` 属性模拟模型特征。

```

public class DataBinding : System.Collections.ArrayList
{
    public DataBinding()
    {
    }
    public void AddNew(int index, object obj)
    {
        this.Add(obj);
    }
}
DataBinding s_model = new DataBinding();
DataBinding i_model = new DataBinding();
System.Windows.Forms.ListBox listBox1 = new System.Windows.Forms.ListBox();
public void SetModel(int whichModel)
{
    if (whichModel==0)
    {
        i_model.Add("i_model Value1");
        i_model.Add("i_model Value2");
        listBox1.DataSource = i_model;
    }
    else
    {
        s_model.Add("s_model Value1");
        s_model.Add("s_model Value2");
        listBox1.DataSource = s_model;
    }
}

```

请参见

概念

[Java Swing 应用程序的转换](#)

com.ms.wfc.data.Connection 示例

下面的示例使用 **com.ms.wfc.data.Connection** 方法的方法:

1. 添加对 Microsoft ActiveX Data Objects (ADO) 2.7 Library (Adodb.dll) 的引用。
2. 用 Connection 对象(**ConnectionClass** 类)建立与数据库的连接。
 - 调用 **ConnectionClass.Open** 方法。

Provider 参数指示 Jet 4 OLE DB 提供程序, 而 *Data Source* 参数指向数据库的物理位置。

[请参见](#)

[其他资源](#)

[ADO API Reference](#)

如何：设置远程处理的安全级别

在 .NET Framework 中，分布式通信（远程）的默认安全级别为 **low**。这会影响向远程方法传递用户定义的对象类型。在某些情况下，可能需要将默认安全级别调整为 **full** 以便允许对象序列化。

调整安全级别

- 修改在转换中生成的配置文件。
- 或 -
- 编辑转换后的代码。

只有在应用程序运行并引发具有以下描述的 [System.Runtime.Serialization.SerializationException](#) 后，才能明确是否需要进行此调整：

由于安全限制，无法访问类型 [System.Runtime.Remoting.ObjRef](#)。

请参见

概念

[CORBA 应用程序的转换](#)

[RMI 应用程序的转换](#)

按包列出的 JLCA 诊断消息

本节包含指向 Java Language Conversion Assistant 的所有诊断消息的链接。包含链接的页是按包进行组织的。

[Java 语言错误、警告和问题](#)

[Com.ms.activex 错误信息](#)

[Com.ms.awt 错误信息](#)

[Com.ms.com 错误信息](#)

[Com.ms.directx 错误信息](#)

[Com.ms.dll 错误信息](#)

[Com.ms.dxmedia 错误信息](#)

[Com.ms.fx 错误信息](#)

[Com.ms.io 错误信息](#)

[Com.ms.jdbc.odbc 错误信息](#)

[Com.ms.lang 错误信息](#)

[Com.ms.mtx 错误信息](#)

[Com.ms.object 错误信息](#)

[Com.ms.ui 错误信息](#)

[Com.ms.util 错误信息](#)

[Com.ms.wfc 错误信息](#)

[Com.ms.wfc.app 错误信息](#)

[Com.ms.wfc.ax 错误信息](#)

[Com.ms.wfc.core 错误信息](#)

[Com.ms.wfc.data 错误信息](#)

[Com.ms.wfc.data.adodb 错误信息](#)

[Com.ms.wfc.data.ui 错误信息](#)

[Com.ms.wfc.io 错误信息](#)

[Com.ms.wfc.ole32 错误信息](#)

[Com.ms.wfc.ui 错误信息](#)

[Com.ms.wfc.util 错误信息](#)

[Com.ms.wfc.win32 错误信息](#)

[Com.ms.win32 错误信息](#)

[Com.sun.image.codec 错误信息](#)

[Java.applet 错误信息](#)

[Java.awt 错误信息](#)

[Java.awt.color 错误信息](#)

[Java.awt.datatransfer 错误信息](#)

[Java.awt.dnd 错误信息](#)

[Java.awt.event 错误信息](#)

[Java.awt.font 错误信息](#)

[Java.awt.geom 错误信息](#)

[Java.awt.im 错误信息](#)

[Java.awt.image 错误信息](#)

[Java.awt.peer 错误信息](#)

[Java.awt.print 错误信息](#)

[Java.beans 错误信息](#)

[Java.io 错误信息](#)

[Java.lang 错误信息](#)

[Java.math 错误信息](#)

[Java.net 错误信息](#)

[Java.rmi 错误信息](#)

[Java.security 错误信息](#)

[Java.sql 错误信息](#)

[Java.text 错误信息](#)

[Java.text.resources 错误信息](#)

[Java.util 错误信息](#)

[Java.util.cab 错误信息](#)

[Java.util.jar 错误信息](#)

[Java.util.zip 错误信息](#)

[Javax.accessibility 错误信息](#)

[Javax.crypto 错误信息](#)

[Javax.ejb 错误信息](#)

[Javax.jms 错误信息](#)

[Javax.mail 错误信息](#)

[Javax.naming 错误信息](#)

[Javax.rmi 错误信息](#)

[Javax.security 错误信息](#)

[Javax.servlet 错误信息](#)

[Javax.servlet.http 错误信息](#)

[Javax.servlet.jsp 错误信息](#)

[Javax.sound 错误信息](#)

[Javax.sql 错误信息](#)

[Javax.swing 错误信息](#)

[Javax.swing.beaninfo 错误信息](#)

[Javax.swing.border 错误信息](#)

[Javax.swing.colorchooser 错误信息](#)

[Javax.swing.event 错误信息](#)

[Javax.swing.filechooser 错误信息](#)

[Javax.swing.table 错误信息](#)

[Javax.swing.text 错误信息](#)

[Javax.swing.tree 错误信息](#)

[Javax.swing.undo 错误信息](#)

[Javax.transaction 错误信息](#)

[Javax.xml 错误信息](#)

[Org.omg 错误信息](#)

[Org.w3c 错误信息](#)

[Org.xml 错误信息](#)

Java 语言错误、警告和问题

- (1002) 编译错误: 无效的语句
- (1003) 注意: 声明已从 Final 更改为变量
- (1005) 注意: 初始化表达式被移动到方法或构造函数中
- (1011) 注意: 生成了新的代码元素
- (1012) 注意: 不支持带标记的 break 语句
- (1014) 注意: 标记语句已被 Java Language Conversion Assistant 移动
- (1015) 注意: 不支持带标记的 Continue 语句
- (1019) 注意: 未随封闭实例提供内部类
- (1021) 注意: 不支持内部类成员字段 this\$0
- (1022) 注意: 不支持局部类声明
- (1023) 注意: Final 变量已作为成员字段复制到内部类中
- (1024) 全局说明: 不支持匿名类声明
- (1025) 注意: 在 C# 中不允许在接口内嵌套接口
- (1027) 注意: 在 .NET Framework 中不支持同步方法
- (1042) 全局警告: Visual C# 中的数据类型可能不同
- (1043) 任务: 方法或属性在 .NET Framework 中返回一个不同的值
- (1045) 注意: 在 .NET Framework 中接口不能包含字段
- (1062) 任务: 通过使用 OleDbDataReader.IsDBNull 可以检查数据库列的 NULL 值
- (1063) 任务: 修改连接字符串以匹配 .NET Framework 格式
- (1064) 任务: 修改字符串连接以匹配 .NET Framework 格式, 并在其中添加 java.util.Properties 信息
- (1066) 任务: setLoginTimeout 必须是 OleDbConnection 构造函数的连接字符串的参数
- (1075) 注意: Font 构造函数可能生成不等效的实例
- (1077) 任务: 不支持类的继承
- (1078) 任务: 确保在此类中使用的资源是有效的资源文件
- (1079) 编译错误: 必须实现从抽象类继承的所有方法
- (1083) 注意: 位数组的大小必须相同才能允许此操作
- (1086) 编译错误: 无法将等效字段分配给整型变量
- (1088) 任务: 没有为 .NET 结构定义 NULL 值
- (1089) 任务: 配置设置的访问和文件格式不同
- (1091) 运行时警告: 等效的构造函数具有不同数目的参数
- (1092) 任务: 参数的值或类型与预期的值或类型不符
- (1093) 任务: 更改代码以访问 .NET 类成员的值
- (1095) 任务: 当前版本中未映射代码元素
- (1096) 注意: 映射到枚举的字段已用整型值初始化
- (1099) 注意: 方法引发的异常可能不同
- (1100) 注意: try 语句中的多个 catch 子句具有同一异常
- (1101) 运行时警告: 映射的参数可能引发异常

- (1102) 编译错误:必须修改类才能调用超类
- (1103) 编译错误:字段有不同的类型
- (1104) 任务:使用 ConnectionString 属性设置登录和密码
- (1105) 任务:不必要的方法调用
- (1107) 全局说明:转换 DataGrid 类的 setAllowAddNew、setAllowDelete 和 setAllowUpdate
- (1108) 任务:在调用之前必须设置或调用属性或方法
- (1109) 编译错误:转换后的参数与方法的参数类型不兼容
- (1113) 编译错误:成员已保留以供程序包在内部使用
- (1114) 任务:必须更改 OpenFileDialog 和 SaveFileDialog 类中的 setFilter 的参数
- (1115) 全局警告:不安全代码需要有编译器开关才能编译
- (1116) 任务:COM 自定义封送处理程序可能变为无效
- (1118) 任务:复选框在分组时不提供互斥机制
- (1119) 注意:用户控件事件属性无效
- (1123) 注意:类映射到接口
- (1124) 注意:已从方法体中移除代码
- (1128) 编译错误:finally 子句级别上的 Break 语句无效
- (1130) 任务:java.sql.DatabaseMetaData 的等效项在 System.Data.DataTable 中处理查询结果
- (1132) 任务:对于格式错误的 URL, 等效类不引发异常
- (1133) 全局警告:Equals 方法可能返回不同的值
- (1135) 全局错误:必须为主窗口调用 System.Windows.Forms.Application.Run 方法
- (1137) 任务:转换为属性的用户 get 或 set 方法有名称冲突
- (1139) 全局警告:未转换目录
- (1140) 全局警告:线程名只能设置一次
- (1141) 注意:Java Language Conversion Assistant 已将来自 COM coclass 的 Java 可调用包装类的代码注释掉
- (1142) 任务:需要重载事件方法
- (1143) 任务:方法不是重写方法
- (1144) 运行时警告:向 componentHidden 中添加额外逻辑
- (1145) 运行时警告:向 componentShown 中添加额外逻辑
- (1146) 任务:无法转换不同行中类实例的声明和结构
- (1147) 任务:<classname> 类被标记为“Sealed”
- (1148) 注意:Synchronized 关键字已被移除
- (1151) 任务:无法访问的构造函数
- (1152) 任务:必须手动转换对注册表的访问
- (1153) 任务:未翻译的语句
- (1154) 注意:类 com.ms.wfc.data.ui.DataNavigator 的等效项的构造函数的参数
- (1155) 任务:从 Delegate 或 MulticastDelegate 派生的类
- (1156) 编译错误:错误的语句
- (1157) 编译错误:方法具有相同的参数类型
- (1158) 任务:已修改访问文件的方式
- (1159) 全局警告:存档路径依赖于工作目录

- (1160) 任务:一个接口包含另一个接口的功能
- (1167) 注意:getParameter 调用不使用字符串
- (1168) 任务:要求重命名参数的 getParameter
- (1169) 注意:属性的默认值已更改
- (1170) 全局警告:类映射到一个结构;该结构不能包含或返回空值
- (1171) 注意:getSource 方法必须处于事件处理例程中
- (1172) 全局错误:输入项目文件已损坏
- (1173) 任务:十六进制文本返回不同的值
- (1174) 注意:在 .NET Framework 中加载二进制文件的方式不同
- (1175) 全局警告:检测到 unsigned long 类型
- (1177) 运行时警告:接口的实现已转换为单个类
- (1178) 任务:方法需要修改
- (1179) 全局警告:请求 OleDbManager 连接字符串
- (1180) 注意:仅支持一个事务性属性
- (1181) 运行时警告:在 C# 代码中表达式被使用多次
- (1182) 任务:会话参数是按其他顺序存储的
- (1183) 运行时警告:Init 功能已经转换
- (1184) 任务:<Target Name> 要求更改接收实例的类型
- (1185) 任务:列访问是通过另一种方式获得的
- (1186) 编译错误:类层次结构差异可能导致编译错误
- (1187) 运行时警告:每次加载 ASP.NET 页时都可能执行 <Member type>
- (1188) 注意:可为转换的 RMI 方法选择一个协议
- (1189) 任务:"多目标消息传送"需要特殊的目标格式设置
- (1190) 任务:远程对象的激活
- (1191) 注意:实例字段已转换为静态
- (1192) 任务:序列化代码可能不能正确转换
- (1193) 编译错误:未在方法转换中使用序列化代码
- (1194) 运行时警告:序列化值的键只能使用一次并且必须与相应的键匹配
- (1195) 任务:更改非虚拟方法的重写修饰符
- (1196) 注意:远程接口文件转换
- (1197) 任务:用函数名的字符串值实例化的委托
- (1198) 注意:继承 servlet 类可能导致错误的变量操作
- (1199) 注意:已合并相应的 javadoc 注释
- (1200) 注意:基类共享的字段已用 new 修饰符进行了重新声明
- (1201) 任务:多通道注册
- (1202) 任务:引用转换需要用户的修改
- (1203) 注意:使用类接口来公开公共成员
- (1204) 注意:访问修饰符已更改
- (1205) 任务:用包装类类型参数重载的具有基元类型参数的方法
- (1206) 全局说明:已转换的项目必须先进行编译才能在 Visual Studio 设计器视图中打开

- (1207) 任务:从 COM 对象扩展的类型应重写所有方法
- (1208) 全局说明:System.Data.OleDb 命名空间可以替换为另一个命名空间
- (1209) 注意:方法未转换为属性
- (1210) 全局警告:可能必须更改反序列化级别
- (1211) 运行时警告:通道注册可能会在没有任何通知的情况下失败
- (1212) 注意:数组界限未经验证
- (1213) 全局说明:在转换过程中添加了配置文件
- (1214) 编译错误:服务器地址必须由用户完成
- (1215) 编译错误:服务器端口必须由用户完成
- (1216) 编译错误:远程对象名必须由用户完成
- (1217) 全局警告:不支持 Java 以外的语言
- (1218) 全局警告:Java 语言非本机定义已转换为 Visual C# 本机构造
- (1219) 全局说明:System.MarshalByRefObject.InitializeLifetimeService 已重写
- (1220) 任务:必须实现 Reset 方法
- (1221) 任务:来自小程序的方法中的父调用
- (1222) 运行时警告:等效构造函数有更多的参数
- (1223) 全局警告:双向语言的输出
- (1224) 任务:方法返回不同的类型
- (1225) 全局警告:基元类型强制转换具有不同的行为
- (1227) 注意:为可见布局复制的代码
- (1228) 注意:注释代码被移到 InitializeComponent 方法
- (1230) 运行时警告:在其他窗体之前添加一个 ContainerControl
- (1231) 任务:多个 Java 类成员被转换为 Visual C# 中的同一个成员
- (1232) 任务:必须实现方法或属性才能保留类逻辑
- (1233) 注意:在转换后的 servlet 类中关键字“this”的行为可能不同
- (1234) 注意:内部类现在可序列化
- (1235) 编译错误:.NET Framework 中没有等效的基类方法
- (1236) 任务:验证 servlet 筛选器列表
- (1237) 注意:EJB 已转换为服务组件
- (1238) 任务:不支持 EJB 回调方法
- (1239) 全局警告:类或接口由多个 EJB 使用
- (1240) 任务:为具有容器托管持久性的 EJB beans 重新实现数据绑定
- (1241) 任务:移动并改编 EJBHome 接口方法
- (1242) 任务:重新实现 EJB 2.0 CMP 查找程序方法
- (1243) 任务:重新实现 EJB 1.1 CMP 查找程序方法
- (1244) 任务:重新实现 EJB 2.0 CMP 关系
- (1245) 任务:在方法级别不支持 EJB 事务属性
- (1246) 任务:不支持 EJB 事务属性
- (1247) 全局警告:事务上下文中引发的异常总是回滚当前事务
- (1248) 警告:不能排除 EJB 方法

- (1249) 警告:不能对未检查的方法禁用安全访问检查
- (1250) 全局警告:需要更改安全级别
- (1251) 注意:自定义封送处理和取消封送处理实现被忽略
- (1252) 全局警告:系统包含文件被忽略
- (1253) 任务:替换 EJB 资源引用
- (1254) 任务:用 .NET 等效项替换 EJB 引用
- (1255) 全局说明:EJB 环境项必须可由当前组件使用
- (1256) 全局说明:不支持 EJB 重入
- (1257) 全局警告:JNDI 安全性
- (1258) 任务:手动调整远程处理上下文的初始化
- (1259) 任务:远程对象注册仅使用默认构造函数
- (1260) 任务:Visual C# 中不需要 IDL 值类型默认工厂
- (1261) 注意:运行库不会调用 Servlet Destroy 方法
- (1262) 任务:不可用类型或包的导入声明可能会导致编译错误
- (1263) 任务:Web 应用程序筛选器的行为不同。
- (1264) 全局警告:.NET Framework 中的事件有不同的行为
- (1265) 运行时警告:等效方法的参数较少
- (1266) 运行时警告:从 System.MarshalByRefObject 派生的对象只能在它们自己的域中被序列化
- (1267) 注意:用于可视化布局的部分代码被移到 InitializeComponent 中
- (1268) 注意:为可序列化的类添加了无参数构造函数
- (1269) 任务:在 Visual C# 中具有多个等效项的类成员可能不起作用
- (1270) 任务:需要替换构造函数
- (1271) 注意:可以移除此方法
- (1272) 任务:未转换不受支持的 IDL 预处理器指令
- (1273) 注意:未转换上下文
- (1274) 全局警告:成员的执行顺序可能不同
- (1275) 注意:RMI 服务器应用程序在终止之前等待客户端请求
- (1277) 任务:等效类不是可序列化的类
- (1278) 注意:在方法返回之前必须给输出参数赋值
- (1279) 运行时警告:传递给委托创建的方法名称可能无效
- (1280) 任务:已找到第三方程序包, 但不必转换
- (1281) 全局警告:由于无效的 XML 字符, 可能发生远程处理异常
- (1282) 全局警告:Visual C# 可能不会正确解析不明确的命名空间
- (1283) 全局警告:移动已转换的 ASP 项目或 JSP 小程序
- (1284) 注意:getModifiers 方法模拟对某些控件可能无效
- (1285) 全局警告:转换的属性编辑器实现可能具有不同的行为
- (1286) 注意:值类型的实现被移动到共享程序集
- (1287) 任务:可能不支持转换字符串
- (1288) 注意:处理密钥的加密类具有不同的行为
- (1290) 全局警告:必须安装 DirectX 以支持声音

- (1291) 任务:结构在 Visual C# 中没有 NULL 的等效项
 - (1292) 注意:用户界面的滚动字段对滚动条转换不起作用
 - (1293) 全局警告:所有 JNDI 异常都映射到 System.Exception
 - (1294) 编译错误:不支持 com.ms.wfc.html 包
 - (1295) 任务:Visual C# 没有以对象为参数的构造函数
 - (1296) 注意:访问修饰符中的更改
 - (1297) 全局警告:必须先启动枚举数, 然后才能访问枚举数的数据
 - (1298) 注意:BeanInfo 功能已转移到相应的 Visual C# 组件中
 - (1299) 任务:必须手动转换返回的表达式
 - (1300) 任务:TypeConverter 需要一个合适的 System.ComponentModel.Design.Serialization.InstanceDescriptor
 - (1301) 全局警告:JavaBeans 源必须位于项目目录
 - (1302) 全局错误:ValueBase 的转换可能导致类型不匹配
 - (1303) 注意:ref 关键字已添加到结构参数中
 - (1304) 注意:JavaBeans 的转换可能会生成部署问题
 - (1305) 全局警告:AwtUI 组件的事件处理程序链接可能丢失
 - (1306) 注意:添加了方法或属性的实现
 - (1307) 任务:不能将语句移动到 InitializeComponent
 - (1308) 注意:已转换侦听器中的方法, 但未使用它
 - (1309) 注意:委托可能有不同的返回值
 - (1310) 任务:EJB 类有多重继承问题
 - (1311) 任务:可能的 EJB 缺少部署说明符
 - (1312) 运行时警告:Mandatory 事务属性没有等效项
 - (1313) 注意:没有被引用的语句是不必要的
 - (1314) 全局警告:转换后的 EJB 启用了默认安全功能
 - (1315) 任务:参数不能通过引用传递
 - (1316) 全局警告:没有应用 EJB 事务属性
 - (1317) 编译错误:不能给 ValueType 数组赋值、对其进行比较或将之作为参数传递
 - (1318) 全局警告:您需要重命名 JAAS 配置文件
- (2001) 运行时警告:在包含文件时, 可能重复了某些必须唯一的服务器端标记
 - (2002) 运行时警告:Aspx 中不允许动态文件包含
 - (2003) 注意:在 ASP.NET 中属性“buffer”只有两个有效值
 - (2004) 注意:请求范围已更改为会话范围
 - (2005) 全局错误:Web.Config customErrors Mode 属性必须设置为“On”
 - (2006) 任务:将不转换文件
 - (2007) 注意:在将自定义标记中的块移动到方法中时, 脚本变量可能会超出范围
 - (2008) 注意:移动自定义标记中的代码块时, 控制结构将无法正常工作
- (4010) 全局错误:ActiveX 引用不同
 - (4011) 全局错误:在文件中找到无效字符
 - (4012) 全局警告:代码页在系统中不可用
 - (4013) 注意:未能找到指定的 TLD 文件, 或未能对其进行分析

(4014) 全局警告:类型库与注册的版本不同

(4015) 注意:用户控件 ID 已更改

(4020) 全局错误:未能解压缩存档文件

(4021) 全局错误:未能读取存档文件

(5000) 任务:方法没有被标记为“Virtual”

请参见

其他资源

[按包列出的 JLCA 诊断消息](#)

[将 Java 应用程序转换为 Visual C#](#)

Com.ms.activex 错误信息

未能转换 com.ms.activex

未能转换 com.ms.activex.ActiveXControl

未能转换 com.ms.activex.ActiveXControlListener

未能转换 com.ms.activex.ActiveXControlServices

未能转换 com.ms.activex.ActiveXInputStream

未能转换 com.ms.activex.ActiveXOutputStream

未能转换 com.ms.activex.ActiveXToolkit

未能转换 com.ms.activex.PropertyDialogThread

Com.ms.awt 错误信息

未能转换 com.ms.awt.AccessibleWrapper
未能转换 com.ms.awt.AWTFinalizable
未能转换 com.ms.awt.AWTFinalizer
未能转换 com.ms.awt.AWTPermission.check
未能转换 com.ms.awt.AWTPermission.pid
未能转换 com.ms.awt.CaretX
未能转换 com.ms.awt.CharSetString
未能转换 com.ms.awt.CharToByteSymbol
未能转换 com.ms.awt.ColorX.ColorX(float, float, float)
未能转换 com.ms.awt.ColorX.ColorX(int)
未能转换 com.ms.awt.ColorX.ColorX(int, int, int)
未能转换 com.ms.awt.ColorX.getHilight
未能转换 com.ms.awt.ColorX.getShadow
未能转换 com.ms.awt.Device.Device
未能转换 com.ms.awt.Device.getBasics
未能转换 com.ms.awt.Device.getDisplayContext
未能转换 com.ms.awt.DrawingSurface
未能转换 com.ms.awt.DrawingSurfaceInfo
未能转换 com.ms.awt.EventFilterListener
未能转换 com.ms.awt.FocusEvent.getOtherComponent
未能转换 com.ms.awt.FocusingTextField
未能转换 com.ms.awt.FontDescriptor
未能转换 com.ms.awt.FontMetricsX.bytesWidth
未能转换 com.ms.awt.FontMetricsX.CHAR_KERNING
未能转换 com.ms.awt.FontMetricsX.charsWidth
未能转换 com.ms.awt.FontMetricsX.FontMetricsX
未能转换 com.ms.awt.FontMetricsX.getAveCharWidth
未能转换 com.ms.awt.FontMetricsX.getFontFace
未能转换 com.ms.awt.FontMetricsX.getFontMetrics
未能转换 com.ms.awt.FontMetricsX.getLeading
未能转换 com.ms.awt.FontMetricsX.getMaxAdvance
未能转换 com.ms.awt.FontMetricsX.getWidths
未能转换 com.ms.awt.FontMetricsX.stringWidth
未能转换 com.ms.awt.FontX.chooseFont
未能转换 com.ms.awt.FontX.EMBEDDED
未能转换 com.ms.awt.FontX.getAttributeList
未能转换 com.ms.awt.FontX.getFlags

未能转换 com.ms.awt.FontX.getFlagsVal
未能转换 com.ms.awt.FontX.getFont
未能转换 com.ms.awt.FontX.getFontList
未能转换 com.ms.awt.FontX.getFontNativeData
未能转换 com.ms.awt.FontX.getNativeData
未能转换 com.ms.awt.FontX.getStyleVal
未能转换 com.ms.awt.FontX.isTypeable
未能转换 com.ms.awt.FontX.matchFace
未能转换 com.ms.awt.FontX.OUTLINE
未能转换 com.ms.awt.FontX.USEDFONT
未能转换 com.ms.awt.GenericEvent
未能转换 com.ms.awt.GraphicsX.bitBlt
未能转换 com.ms.awt.GraphicsX.cng
未能转换 com.ms.awt.GraphicsX.comp
未能转换 com.ms.awt.GraphicsX.copyArea
未能转换 com.ms.awt.GraphicsX.drawBezier
未能转换 com.ms.awt.GraphicsX.drawChars
未能转换 com.ms.awt.GraphicsX.drawCharsWithoutFxFont
未能转换 com.ms.awt.GraphicsX.drawOutlineChar
未能转换 com.ms.awt.GraphicsX.drawOutlinePolygon
未能转换 com.ms.awt.GraphicsX.drawPixels
未能转换 com.ms.awt.GraphicsX.drawRoundRect
未能转换 com.ms.awt.GraphicsX.drawScanLines
未能转换 com.ms.awt.GraphicsX.drawT2Curve
未能转换 com.ms.awt.GraphicsX.fill3DRect
未能转换 com.ms.awt.GraphicsX.fillRoundRect
未能转换 com.ms.awt.GraphicsX.gdc
未能转换 com.ms.awt.GraphicsX.getColorType
未能转换 com.ms.awt.GraphicsX.getGlyphOutline
未能转换 com.ms.awt.GraphicsX.go
未能转换 com.ms.awt.GraphicsX.GraphicsX
未能转换 com.ms.awt.GraphicsX.image
未能转换 com.ms.awt.GraphicsX.originX
未能转换 com.ms.awt.GraphicsX.originY
未能转换 com.ms.awt.GraphicsX.resetSurfaceParams
未能转换 com.ms.awt.GraphicsX.setClip
未能转换 com.ms.awt.GraphicsX.setPaintMode
未能转换 com.ms.awt.GraphicsX.setSurfaceOffset
未能转换 com.ms.awt.GraphicsX.setSurfaceOwner
未能转换 com.ms.awt.GraphicsX.setSurfaceVisRgn

未能转换 com.ms.awt.GraphicsX.setXORMode
未能转换 com.ms.awt.GraphicsXConstants.BDR_VALID
未能转换 com.ms.awt.HeavyComponent
未能转换 com.ms.awt.HorizBagLayout
未能转换 com.ms.awt.ImageX
未能转换 com.ms.awt.ListLayout
未能转换 com.ms.awt.MenuBarX.getItemId
未能转换 com.ms.awt.MenuBarX.MenuBarX(int)
未能转换 com.ms.awt.MenuBarX.MenuBarX(int, Applet, String)
未能转换 com.ms.awt.MenuBarX.MenuBarX(int, String)
未能转换 com.ms.awt.MenuItemX.addNotify
未能转换 com.ms.awt.MenuItemX.getId
未能转换 com.ms.awt.MenuX.CheckMenuItem
未能转换 com.ms.awt.MenuX.getItemId
未能转换 com.ms.awt.MenuXConstants
未能转换 com.ms.awt.OrientableFlowLayout
未能转换 com.ms.awt.PhysicalDrawingSurface
未能转换 com.ms.awt.PlatformFont
未能转换 com.ms.awt.VariableGridLayout
未能转换 com.ms.awt.VerticalBagLayout
未能转换 com.ms.awt.WClipboard.lostClipboard
未能转换 com.ms.awt.WClipboard.lostSelectionOwnership
未能转换 com.ms.awt.WClipboard.setToolkit
未能转换 com.ms.awt.WClipboard.WClipboard
未能转换 com.ms.awt.WComponentPeer
未能转换 com.ms.awt.WDragSession
未能转换 com.ms.awt.WEventQueue
未能转换 com.ms.awt.WFileDialogPeer
未能转换 com.ms.awt.WGuiCallback
未能转换 com.ms.awt.WHeavyPeer
未能转换 com.ms.awt.Win32SystemResourceDecoder
未能转换 com.ms.awt.WinEvent.notify
未能转换 com.ms.awt.WPrintGraphics
未能转换 com.ms.awt.WPrintJob
未能转换 com.ms.awt.WToolkit

Com.ms.com 错误信息

未能转换 com.ms.com.AnsiStringMarshaller
未能转换 com.ms.com.AnsiStringRef
未能转换 com.ms.com.COAUTHIDENTITY.cbByValSize
未能转换 com.ms.com.COAUTHIDENTITY.domain
未能转换 com.ms.com.COAUTHIDENTITY.flags
未能转换 com.ms.com.COAUTHIDENTITY.fromPtr
未能转换 com.ms.com.COAUTHIDENTITY.password
未能转换 com.ms.com.COAUTHIDENTITY.toExternal
未能转换 com.ms.com.COAUTHIDENTITY.toJava
未能转换 com.ms.com.COAUTHIDENTITY.toPtr
未能转换 com.ms.com.COAUTHIDENTITY.user
未能转换 com.ms.com.COAUTHINFO.pAuthIdentityData
未能转换 com.ms.com.COAUTHINFO.pwszServerPrincName
未能转换 com.ms.com.ComContext
未能转换 com.ms.com.ComException.ComException
未能转换 com.ms.com.ComException.getHelpContext
未能转换 com.ms.com.ComException.getHResult
未能转换 com.ms.com.ComException.hr
未能转换 com.ms.com.ComException.m_helpContext
未能转换 com.ms.com.ComException.m_helpFile
未能转换 com.ms.com.ComException.m_source
未能转换 com.ms.com.ComFailException.ComFailException
未能转换 com.ms.com.ComLib.ComLib
未能转换 com.ms.com.ComLib.declareMessagePumpThread
未能转换 com.ms.com.ComLib.executeOnContext
未能转换 com.ms.com.ComLib.freeUnusedLibraries
未能转换 com.ms.com.ComLib.IENVNextMarshalerC2J
未能转换 com.ms.com.ComLib.IENVNextMarshalerJ2C
未能转换 com.ms.com.ComLib.IID_IDispatch
未能转换 com.ms.com.ComLib.IID_IUnknown
未能转换 com.ms.com.ComLib.isEqualUnknown
未能转换 com.ms.com.ComLib.jcdwClassOffsetOf
未能转换 com.ms.com.ComLib.jcdwOffsetOf
未能转换 com.ms.com.ComLib.makeProxyRef
未能转换 com.ms.com.ComLib.ownsCleanup
未能转换 com.ms.com.ComLib.ptrToUnknown
未能转换 com.ms.com.ComLib.setDataWrapperSize

未能转换 com.ms.com.ComLib.startMTAThread
未能转换 com.ms.com.ComLib.supportsInterface
未能转换 com.ms.com.ComLib.threadStartMTA
未能转换 com.ms.com.ComLib.unknownToPtr
未能转换 com.ms.com.ComSuccessException.ComSuccessException
未能转换 com.ms.com.CONNECTDATA.pUnk
未能转换 com.ms.com.COSERVERINFO.pAuthInfo
未能转换 com.ms.com.COSERVERINFO.pwszName
未能转换 com.ms.com.CUnknown
未能转换 com.ms.com.CustomLib
未能转换 com.ms.com.Dispatch
未能转换 com.ms.com.DispatchProxy
未能转换 com.ms.com.Generic
未能转换 com.ms.com.Guid.Guid
未能转换 com.ms.com.Guid.setByStr
未能转换 com.ms.com.IAccessible.iid
未能转换 com.ms.com.IAccessibleDefault.iid
未能转换 com.ms.com.IBindCtx.iid
未能转换 com.ms.com.IBindCtx.RegisterObjectBound
未能转换 com.ms.com.IBindCtx.RevokeObjectBound
未能转换 com.ms.com.IBindCtx.RevokeObjectParam
未能转换 com.ms.com.IClassFactory
未能转换 com.ms.com.IClassFactory2
未能转换 com.ms.com.IConnectionPoint.Advise
未能转换 com.ms.com.IConnectionPoint.iid
未能转换 com.ms.com.IConnectionPointContainer.iid
未能转换 com.ms.com.IEnumConnectionPoints.iid
未能转换 com.ms.com.IEnumConnectionPoints.Next
未能转换 com.ms.com.IEnumConnections.iid
未能转换 com.ms.com.IEnumConnections.Next
未能转换 com.ms.com.IEnumConnections.Skip
未能转换 com.ms.com.IEnumMoniker.iid
未能转换 com.ms.com.IEnumMoniker.Next
未能转换 com.ms.com.IEnumSTATSTG
未能转换 com.ms.com.IEnumString.iid
未能转换 com.ms.com.IEnumString.Next
未能转换 com.ms.com.IEnumUnknown
未能转换 com.ms.com.IEnumVariant.Clone
未能转换 com.ms.com.IExternalConnectionSink
未能转换 com.ms.com.IIDIsMarshaler

未能转换 com.ms.com.ILicenseMgr
未能转换 com.ms.com.ILockBytes
未能转换 com.ms.com.IMarshal.GetMarshalSizeMax
未能转换 com.ms.com.IMarshal.GetUnmarshalClass
未能转换 com.ms.com.IMarshal.iid
未能转换 com.ms.com.IMoniker.BindToObject
未能转换 com.ms.com.IMoniker.iid
未能转换 com.ms.com.IMoniker.IsDirty
未能转换 com.ms.com.IMoniker.AreEqual
未能转换 com.ms.com.IMoniker.IsRunning
未能转换 com.ms.com.IParseDisplayName
未能转换 com.ms.com.IPersist.iid
未能转换 com.ms.com.IPersistFile.iid
未能转换 com.ms.com.IPersistFile.IsDirty
未能转换 com.ms.com.IPersistFile.Save
未能转换 com.ms.com.IPersistStorage
未能转换 com.ms.com.IPersistStream.GetClassID
未能转换 com.ms.com.IPersistStream.iid
未能转换 com.ms.com.IPersistStreamInit
未能转换 com.ms.com.IPropertyNotifySink
未能转换 com.ms.com.IROTData
未能转换 com.ms.com.IRunningObjectTable.GetObject
未能转换 com.ms.com.IRunningObjectTable.GetTimeOfLastChange
未能转换 com.ms.com.IRunningObjectTable.iid
未能转换 com.ms.com.IRunningObjectTable.IsRunning
未能转换 com.ms.com.IRunningObjectTable.NoteChangeTime
未能转换 com.ms.com.IRunningObjectTable.Register
未能转换 com.ms.com.ISequentialStream
未能转换 com.ms.com.ISequentialStream.iid
未能转换 com.ms.com.IServiceProvider
未能转换 com.ms.com.IStorage
未能转换 com.ms.com.IStream.CopyTo
未能转换 com.ms.com.IStream.iid
未能转换 com.ms.com.IStream.LOCK_EXCLUSIVE
未能转换 com.ms.com.IStream.LOCK_ONLYONCE
未能转换 com.ms.com.IStream.LOCK_WRITE
未能转换 com.ms.com.IStream.Read
未能转换 com.ms.com.IStream.Seek
未能转换 com.ms.com.IStream.STATFLAG_DEFAULT
未能转换 com.ms.com.IStream.STATFLAG_NONAME

未能转换 com.ms.com.IStream.STATFLAG_NOOPEN
未能转换 com.ms.com.IStream.STGC_DANGEROUSLYCOMMITMERELEYTODISKCACHE
未能转换 com.ms.com.IStream.STGC_DEFAULT
未能转换 com.ms.com.IStream.STGC_ONLYIFCURRENT
未能转换 com.ms.com.IStream.STGC_OVERWRITE
未能转换 com.ms.com.IStream.STREAM_SEEK_CUR
未能转换 com.ms.com.IStream.STREAM_SEEK_END
未能转换 com.ms.com.IStream.STREAM_SEEK_SET
未能转换 com.ms.com.IStream.Write
未能转换 com.ms.com.LicenseMgr
未能转换 com.ms.com.LICINFO
未能转换 com.ms.com.MULTI_QI
未能转换 com.ms.com.NoAutoMarshaling
未能转换 com.ms.com.NoAutoScripting
未能转换 com.ms.com.SafeArray.destroy
未能转换 com.ms.com.SafeArray.getFeatures
未能转换 com.ms.com.SafeArray.getNumLocks
未能转换 com.ms.com.SafeArray.getPhysicalSafeArray
未能转换 com.ms.com.SafeArray.getvt
未能转换 com.ms.com.SafeArray.reinit
未能转换 com.ms.com.SafeArray.reinterpretType
未能转换 com.ms.com.SizesMarshaler
未能转换 com.ms.com.STATSTG.clsid_data1
未能转换 com.ms.com.STATSTG.clsid_data2
未能转换 com.ms.com.STATSTG.clsid_data3
未能转换 com.ms.com.STATSTG.STGTY_LOCKBYTES
未能转换 com.ms.com.STATSTG.STGTY_PROPERTY
未能转换 com.ms.com.STATSTG.STGTY_STORAGE
未能转换 com.ms.com.STATSTG.STGTY_STREAM
未能转换 com.ms.com.StdCOMClassObject
未能转换 com.ms.com.UniStringMarshall
未能转换 com.ms.com.UniStringRef
未能转换 com.ms.com.Variant.changeType
未能转换 com.ms.com.Variant.clone
未能转换 com.ms.com.Variant.cloneIndirect
未能转换 com.ms.com.Variant.getDate
未能转换 com.ms.com.Variant.getEmpty
未能转换 com.ms.com.Variant.getErrorRef
未能转换 com.ms.com.Variant.getNull
未能转换 com.ms.com.Variant.getVariantArray

未能转换 com.ms.com.Variant.getVariantArrayRef
未能转换 com.ms.com.Variant.getvt
未能转换 com.ms.com.Variant.noParam
未能转换 com.ms.com.Variant.putError
未能转换 com.ms.com.Variant.putSafeArrayRefHelper
未能转换 com.ms.com.Variant.toError
未能转换 com.ms.com.Variant.toScriptObject
未能转换 com.ms.com.Variant.toVariantArray
未能转换 com.ms.com.Variant.Variant
未能转换 com.ms.com.Variant.VariantByref
未能转换 com.ms.com.Variant.VariantClear
未能转换 com.ms.com.Variant.VariantCurrency
未能转换 com.ms.com.Variant.VariantEmpty
未能转换 com.ms.com.Variant.VariantError
未能转换 com.ms.com.Variant.VariantNull
未能转换 com.ms.com.Variant.VariantTypeMask
未能转换 com.ms.com.Variant.VariantVariant

Com.ms.directx 错误信息

未能转换 com.ms.directX.D3dFindDeviceResult.GetGuid
未能转换 com.ms.directX.D3dFindDeviceSearch.getGuid
未能转换 com.ms.directX.D3dFindDeviceSearch.setGuid
未能转换 com.ms.directX.Direct3dRMFace.getVertices
未能转换 com.ms.directX.Direct3dRMMesh.getVertices
未能转换 com.ms.directX.Direct3dRMMesh.setVertices
com.ms.directX.Direct3dRMMeshBuilder.addFaces
com.ms.directX.Direct3dRMMeshBuilder.getVertices
未能转换 com.ms.directX.DirectDraw.createPalette
未能转换 com.ms.directX.DirectDraw.setCooperativeLevel
未能转换 com.ms.directX.DirectDrawClipper.resetSurfaceParams
未能转换 com.ms.directX.DirectDrawClipper.setComponent
未能转换 com.ms.directX.DirectDrawClipper.setSurfaceOffset
未能转换 com.ms.directX.DirectDrawClipper.setSurfaceOwner
未能转换 com.ms.directX.DirectDrawClipper.setSurfaceVisRgn
未能转换 com.ms.directX.DirectDrawPalette.getColorEntries
未能转换 com.ms.directX.DirectDrawPalette.getPaletteEntries
未能转换 com.ms.directX.DirectDrawPalette.setEntries
未能转换 com.ms.directX.DirectSound.createSoundBuffer
未能转换 com.ms.directX.DirectSound.setCooperativeLevel
未能转换 com.ms.directX.DirectSoundBuffer.getFormat
未能转换 com.ms.directX.DirectSoundBuffer.initialize
未能转换 com.ms.directX.DirectSoundBuffer.setFormat
未能转换 com.ms.directX.DirectSoundResource.loadWaveFile
未能转换 com.ms.directX.DirectSoundResource.loadWaveResource
未能转换 com.ms.directX.DirectXConstants
未能转换 com.ms.directX.PaletteEntry
未能转换 com.ms.directX.WaveFormatEx

Com.ms.dll 错误信息

未能转换 com.ms.dll.Callback

未能转换 com.ms.dll.DllLib.addrOf

未能转换 com.ms.dll.DllLib.addrOfPinnedObject

未能转换 com.ms.dll.DllLib.copy

未能转换 com.ms.dll.DllLib.DllLib

未能转换 com.ms.dll.DllLib.freePinnedHandle

未能转换 com.ms.dll.DllLib.getPinnedHandle

未能转换 com.ms.dll.DllLib.getPinnedObject

未能转换 com.ms.dll.DllLib.isStruct

未能转换 com.ms.dll.DllLib.propagateStructFields

未能转换 com.ms.dll.DllLib.resize

未能转换 com.ms.dll.ParameterCountMismatchError

未能转换 com.ms.dll.StringMarshaler

Com.ms.dxmedia 错误信息

未能转换 com.ms.dxmedia.AppTriggeredEvent
未能转换 com.ms.dxmedia.ArrayBvr.ArrayBvr
未能转换 com.ms.dxmedia.ArrayBvr.getCOMPtr
未能转换 com.ms.dxmedia.ArrayBvr.newUninitBvr
未能转换 com.ms.dxmedia.ArrayBvr.setCOMBvr
未能转换 com.ms.dxmedia.Bbox2Bvr.Bbox2Bvr
未能转换 com.ms.dxmedia.Bbox2Bvr.getCOMPtr
未能转换 com.ms.dxmedia.Bbox2Bvr.getMax
未能转换 com.ms.dxmedia.Bbox2Bvr.getMin
未能转换 com.ms.dxmedia.Bbox2Bvr.newUninitBehavior
未能转换 com.ms.dxmedia.Bbox2Bvr.newUninitBvr
未能转换 com.ms.dxmedia.Bbox2Bvr.setCOMBvr
未能转换 com.ms.dxmedia.Bbox3Bvr.Bbox3Bvr
未能转换 com.ms.dxmedia.Bbox3Bvr.getCOMPtr
未能转换 com.ms.dxmedia.Bbox3Bvr.getMax
未能转换 com.ms.dxmedia.Bbox3Bvr.getMin
未能转换 com.ms.dxmedia.Bbox3Bvr.newUninitBehavior
未能转换 com.ms.dxmedia.Bbox3Bvr.newUninitBvr
未能转换 com.ms.dxmedia.Bbox3Bvr.setCOMBvr
未能转换 com.ms.dxmedia.Behavior.Behavior
未能转换 com.ms.dxmedia.Behavior.debug
未能转换 com.ms.dxmedia.Behavior.extract
未能转换 com.ms.dxmedia.Behavior.getCOMBvr
未能转换 com.ms.dxmedia.Behavior.newUninitBehavior
未能转换 com.ms.dxmedia.BooleanBvr.BooleanBvr
未能转换 com.ms.dxmedia.BooleanBvr.getCOMPtr
未能转换 com.ms.dxmedia.BooleanBvr.newUninitBehavior
未能转换 com.ms.dxmedia.BooleanBvr.setCOMBvr
未能转换 com.ms.dxmedia.CallbackNotifier
未能转换 com.ms.dxmedia.CameraBvr.CameraBvr
未能转换 com.ms.dxmedia.CameraBvr.getCOMPtr
未能转换 com.ms.dxmedia.CameraBvr.newUninitBehavior
未能转换 com.ms.dxmedia.CameraBvr.newUninitBvr
未能转换 com.ms.dxmedia.CameraBvr.setCOMBvr
未能转换 com.ms.dxmedia.ColorBvr.ColorBvr
未能转换 com.ms.dxmedia.ColorBvr.getCOMPtr
未能转换 com.ms.dxmedia.ColorBvr.newUninitBehavior

未能转换 com.ms.dxmedia.ColorBvr.NewUninitBvr
未能转换 com.ms.dxmedia.ColorBvr.setCOMBvr
未能转换 com.ms.dxmedia.Cycler
未能转换 com.ms.dxmedia.DashStyleBvr.DashStyleBvr
未能转换 com.ms.dxmedia.DashStyleBvr.getCOMPtr
未能转换 com.ms.dxmedia.DashStyleBvr.newUninitBehavior
未能转换 com.ms.dxmedia.DashStyleBvr.newUninitBvr
未能转换 com.ms.dxmedia.DashStyleBvr.setCOMBvr
未能转换 com.ms.dxmedia.DefaultErrReceiver
未能转换 com.ms.dxmedia.DXMApplet
未能转换 com.ms.dxmedia.DXMCanvas
未能转换 com.ms.dxmedia.DXMCanvasBase
未能转换 com.ms.dxmedia.DXMDestroyCallback
未能转换 com.ms.dxmedia.DXMEvent.DXMEvent
未能转换 com.ms.dxmedia.DXMEvent.getCOMPtr
未能转换 com.ms.dxmedia.DXMEvent.newUninitBehavior
未能转换 com.ms.dxmedia.DXMEvent.registerCallback
未能转换 com.ms.dxmedia.DXMEvent.setCOMBvr
未能转换 com.ms.dxmedia.DXMEception.DXMEception
未能转换 com.ms.dxmedia.EndStyleBvr.EndStyleBvr
未能转换 com.ms.dxmedia.EndStyleBvr.getCOMPtr
未能转换 com.ms.dxmedia.EndStyleBvr.newUninitBehavior
未能转换 com.ms.dxmedia.EndStyleBvr.newUninitBvr
未能转换 com.ms.dxmedia.EndStyleBvr.setCOMBvr
未能转换 com.ms.dxmedia.ErrorAndWarningReceiver
未能转换 com.ms.dxmedia.EventCallbackObject
未能转换 com.ms.dxmedia.FontStyleBvr.FontStyleBvr
未能转换 com.ms.dxmedia.FontStyleBvr.getCOMPtr
未能转换 com.ms.dxmedia.FontStyleBvr.newUninitBehavior
未能转换 com.ms.dxmedia.FontStyleBvr.NewUninitBvr
未能转换 com.ms.dxmedia.FontStyleBvr.setCOMBvr
未能转换 com.ms.dxmedia.GeometryBvr.GeometryBvr
未能转换 com.ms.dxmedia.GeometryBvr.getCOMPtr
未能转换 com.ms.dxmedia.GeometryBvr.newUninitBehavior
未能转换 com.ms.dxmedia.GeometryBvr.setCOMBvr
未能转换 com.ms.dxmedia.ImageBvr.getCOMPtr
未能转换 com.ms.dxmedia.ImageBvr.ImageBvr
未能转换 com.ms.dxmedia.ImageBvr.newUninitBehavior
未能转换 com.ms.dxmedia.ImageBvr.setCOMBvr
未能转换 com.ms.dxmedia.JoinStyleBvr.getCOMPtr

未能转换 com.ms.dxmedia.JoinStyleBvr.JoinStyleBvr
未能转换 com.ms.dxmedia.JoinStyleBvr.newUninitBehavior
未能转换 com.ms.dxmedia.JoinStyleBvr.setCOMBvr
未能转换 com.ms.dxmedia.LineStyleBvr.getCOMPtr
未能转换 com.ms.dxmedia.LineStyleBvr.LineStyleBvr
未能转换 com.ms.dxmedia.LineStyleBvr.newUninitBehavior
未能转换 com.ms.dxmedia.LineStyleBvr.setCOMBvr
未能转换 com.ms.dxmedia.MatteBvr.getCOMPtr
未能转换 com.ms.dxmedia.MatteBvr.MatteBvr
未能转换 com.ms.dxmedia.MatteBvr.newUninitBehavior
未能转换 com.ms.dxmedia.MatteBvr.setCOMBvr
未能转换 com.ms.dxmedia.MicrophoneBvr.getCOMPtr
未能转换 com.ms.dxmedia.MicrophoneBvr.MicrophoneBvr
未能转换 com.ms.dxmedia.MicrophoneBvr.newUninitBehavior
未能转换 com.ms.dxmedia.MicrophoneBvr.setCOMBvr
未能转换 com.ms.dxmedia.Model.cleanup
未能转换 com.ms.dxmedia.Model.createModel
未能转换 com.ms.dxmedia.Model.getImportBase
未能转换 com.ms.dxmedia.Model.modifyPreferences
未能转换 com.ms.dxmedia.Model.receiveInputImages
未能转换 com.ms.dxmedia.Model.setImportBase
未能转换 com.ms.dxmedia.ModelMakerApplet
未能转换 com.ms.dxmedia.ModifiableBehavior
未能转换 com.ms.dxmedia.MontageBvr.getCOMPtr
未能转换 com.ms.dxmedia.MontageBvr.MontageBvr
未能转换 com.ms.dxmedia.MontageBvr.newUninitBehavior
未能转换 com.ms.dxmedia.MontageBvr.setCOMBvr
未能转换 com.ms.dxmedia.NumberBvr.getCOMPtr
未能转换 com.ms.dxmedia.NumberBvr.newUninitBehavior
未能转换 com.ms.dxmedia.NumberBvr.NumberBvr
未能转换 com.ms.dxmedia.NumberBvr.setCOMBvr
未能转换 com.ms.dxmedia.PairObject.PairObject
未能转换 com.ms.dxmedia.Path2Bvr.getCOMPtr
未能转换 com.ms.dxmedia.Path2Bvr.newUninitBehavior
未能转换 com.ms.dxmedia.Path2Bvr.Path2Bvr
未能转换 com.ms.dxmedia.Path2Bvr.setCOMBvr
未能转换 com.ms.dxmedia.PickableGeometry.PickableGeometry
未能转换 com.ms.dxmedia.PickableImage.PickableImage
未能转换 com.ms.dxmedia.Point2Bvr.getCOMPtr
未能转换 com.ms.dxmedia.Point2Bvr.newUninitBehavior

未能转换 com.ms.dxmedia.Point2Bvr.Point2Bvr
未能转换 com.ms.dxmedia.Point2Bvr.setCOMBvr
未能转换 com.ms.dxmedia.Point3Bvr.getCOMPtr
未能转换 com.ms.dxmedia.Point3Bvr.newUninitBehavior
未能转换 com.ms.dxmedia.Point3Bvr.Point3Bvr
未能转换 com.ms.dxmedia.Point3Bvr.setCOMBvr
未能转换 com.ms.dxmedia.Preferences.COLOR_KEY_BLUE
未能转换 com.ms.dxmedia.Preferences.COLOR_KEY_GREEN
未能转换 com.ms.dxmedia.Preferences.COLOR_KEY_RED
未能转换 com.ms.dxmedia.Preferences.DITHERING
未能转换 com.ms.dxmedia.Preferences.ENGINE_OPTIMIZATIONS
未能转换 com.ms.dxmedia.Preferences.FILL_MODE
未能转换 com.ms.dxmedia.Preferences.FILL_MODE_POINT
未能转换 com.ms.dxmedia.Preferences.FILL_MODE_SOLID
未能转换 com.ms.dxmedia.Preferences.FILL_MODE_WIREFRAME
未能转换 com.ms.dxmedia.Preferences.MAX_FRAMES_PER_SEC
未能转换 com.ms.dxmedia.Preferences.OVERRIDE_APPLICATION_PREFERENCES
未能转换 com.ms.dxmedia.Preferences.PERSPECTIVE_CORRECT
未能转换 com.ms.dxmedia.Preferences.RGB_LIGHTING_MODE
未能转换 com.ms.dxmedia.Preferences.SHADE_MODE
未能转换 com.ms.dxmedia.Preferences.SHADE_MODE_FLAT
未能转换 com.ms.dxmedia.Preferences.SHADE_MODE_GOURAUD
未能转换 com.ms.dxmedia.Preferences.SHADE_MODE_PHONG
未能转换 com.ms.dxmedia.Preferences.TEXTURE_QUALITY
未能转换 com.ms.dxmedia.Preferences.TEXTURE_QUALITY_LINEAR
未能转换 com.ms.dxmedia.Preferences.TEXTURE_QUALITY_NEAREST
未能转换 com.ms.dxmedia.Preferences.USE_3D_HW
未能转换 com.ms.dxmedia.Preferences.USE_VIDEOMEM
未能转换 com.ms.dxmedia.PropertyDispatcher
未能转换 com.ms.dxmedia.SoundBvr.getCOMPtr
未能转换 com.ms.dxmedia.SoundBvr.newUninitBehavior
未能转换 com.ms.dxmedia.SoundBvr.setCOMBvr
未能转换 com.ms.dxmedia.SoundBvr.SoundBvr
未能转换 com.ms.dxmedia_Statics.makeBvrFromInterface
未能转换 com.ms.dxmedia_StaticsBase._site
未能转换 com.ms.dxmedia_StaticsBase.BvrHook
未能转换 com.ms.dxmedia_StaticsBase.checkRead
未能转换 com.ms.dxmedia_StaticsBase.cm
未能转换 com.ms.dxmedia_StaticsBase.foot
未能转换 com.ms.dxmedia_StaticsBase.getCOMPtr

未能转换 com.ms.dxmedia_StaticsBase.handleError
未能转换 com.ms.dxmedia_StaticsBase.importGeometry
未能转换 com.ms.dxmedia_StaticsBase.importImage
未能转换 com.ms.dxmedia_StaticsBase.importMovie(异步)
未能转换 com.ms.dxmedia_StaticsBase.importMovie(同步)
未能转换 com.ms.dxmedia_StaticsBase.importSound(异步)
未能转换 com.ms.dxmedia_StaticsBase.importSound(同步)
未能转换 com.ms.dxmedia_StaticsBase.inch
未能转换 com.ms.dxmedia_StaticsBase.meter
未能转换 com.ms.dxmedia_StaticsBase.mm
未能转换 com.ms.dxmedia_StaticsBase.registerErrorAndWarningReceiver
未能转换 com.ms.dxmedia_StaticsBase.unregisterCallback
未能转换 com.ms.dxmedia_StringBvr.getCOMPtr
未能转换 com.ms.dxmedia_StringBvr.newUninitBehavior
未能转换 com.ms.dxmedia_StringBvr.setCOMBvr
未能转换 com.ms.dxmedia_StringBvr.StringBvr
未能转换 com.ms.dxmedia_Transform2Bvr.getCOMPtr
未能转换 com.ms.dxmedia_Transform2Bvr.newUninitBehavior
未能转换 com.ms.dxmedia_Transform2Bvr.setCOMBvr
未能转换 com.ms.dxmedia_Transform2Bvr.Transform2Bvr
未能转换 com.ms.dxmedia_Transform3Bvr.getCOMPtr
未能转换 com.ms.dxmedia_Transform3Bvr.newUninitBehavior
未能转换 com.ms.dxmedia_Transform3Bvr.setCOMBvr
未能转换 com.ms.dxmedia_Transform3Bvr.Transform3Bvr
未能转换 com.ms.dxmedia_TupleBvr.getCOMPtr
未能转换 com.ms.dxmedia_TupleBvr.setCOMBvr
未能转换 com.ms.dxmedia_TupleBvr.TupleBvr
未能转换 com.ms.dxmedia_UntilNotifierCB
未能转换 com.ms.dxmedia_Vector2Bvr.getCOMPtr
未能转换 com.ms.dxmedia_Vector2Bvr.newUninitBehavior
未能转换 com.ms.dxmedia_Vector2Bvr.setCOMBvr
未能转换 com.ms.dxmedia_Vector2Bvr.Vector2Bvr
未能转换 com.ms.dxmedia_Vector3Bvr.getCOMPtr
未能转换 com.ms.dxmedia_Vector3Bvr.newUninitBehavior
未能转换 com.ms.dxmedia_Vector3Bvr.setCOMBvr
未能转换 com.ms.dxmedia_Vector3Bvr.Vector3Bvr
未能转换 com.ms.dxmedia_Viewer.getCurrentTickTime
未能转换 com.ms.dxmedia_Viewer.registerErrorAndWarningReceiver
未能转换 com.ms.dxmedia_Viewer.startModel
未能转换 com.ms.dxmedia_Viewer.tick

Com.ms.fx 错误信息

未能转换 com.ms.fx.BaseColor.BaseColor
未能转换 com.ms.fx.fullTxtRun
未能转换 com.ms.fx.FxBrushPen.drawBytesCallback
未能转换 com.ms.fx.FxBrushPen.drawCharsCallback
未能转换 com.ms.fx.FxBrushPen.drawRoundRectCallback
未能转换 com.ms.fx.FxBrushPen.drawScanLinesCallback
未能转换 com.ms.fx.FxBrushPen.drawStringCallback
未能转换 com.ms.fx.FxBrushPen.fill3DRectCallback
未能转换 com.ms.fx.FxBrushPen.fillRoundRectCallback
未能转换 com.ms.fx.FxCaret
未能转换 com.ms.fx.FxColor.brightenColor
未能转换 com.ms.fx.FxColor.darkenColor
未能转换 com.ms.fx.FxColor.FxColor
未能转换 com.ms.fx.FxColor.getHilight
未能转换 com.ms.fx.FxColor.getShadow
未能转换 com.ms.fx.FxComponentImage
未能转换 com.ms.fx.FxComponentTexture.FxComponentTexture
未能转换 com.ms.fx.FxCurve
未能转换 com.ms.fx.FxEllipse
未能转换 com.ms.fx.FxFONTANTIALIAS
未能转换 com.ms.fx.FxFONT.drawEffects
未能转换 com.ms.fx.FxFONTXFONT
未能转换 com.ms.fx.FxFONT.FxFONT
未能转换 com.ms.fx.FxFONT.getAttributeList
未能转换 com.ms.fx.FxFONT.getEmboldenedFont
未能转换 com.ms.fx.FxFONT.getFontFlags
未能转换 com.ms.fx.FxFONT.getFlagsVal
未能转换 com.ms.fx.FxFONT.getFont
未能转换 com.ms.fx.FxFONT.getFontList
未能转换 com.ms.fx.FxFONT.getStyleVal
未能转换 com.ms.fx.FxFONT.matchFace
未能转换 com.ms.fx.FxFONT.STRIKEOUT
未能转换 com.ms.fx.FxFONT.UNDERLINE
未能转换 com.ms.fx.FxFONT.USEDFONT
未能转换 com.ms.fx.FxFONTMetrics
未能转换 com.ms.fx.FxFONTMetricsOther
未能转换 com.ms.fx.FxFormattedText

未能转换 com.ms.fx.FxGraphicMetaFile.addObjectToTable
未能转换 com.ms.fx.FxGraphicMetaFile.debugging
未能转换 com.ms.fx.FxGraphicMetaFile.enumerate
未能转换 com.ms.fx.FxGraphicMetaFile.FxGraphicMetaFile
未能转换 com.ms.fx.FxGraphicMetaFile.hdc
未能转换 com.ms.fx.FxGraphicMetaFile(hdcStates
未能转换 com.ms.fx.FxGraphicMetaFile.play
未能转换 com.ms.fx.FxGraphicMetaFile.records
未能转换 com.ms.fx.FxGraphicMetaFile.removeObjectFromTable
未能转换 com.ms.fx.FxGraphics.drawBezier
未能转换 com.ms.fx.FxGraphics.drawBorder
未能转换 com.ms.fx.FxGraphics.drawChars
未能转换 com.ms.fx.FxGraphics.drawCharsWithoutFxFont
未能转换 com.ms.fx.FxGraphics.drawOutlineChar
未能转换 com.ms.fx.FxGraphics.drawOutlinePolygon
未能转换 com.ms.fx.FxGraphics.drawPixels
未能转换 com.ms.fx.FxGraphics.drawScanLines
未能转换 com.ms.fx.FxGraphics.drawString
未能转换 com.ms.fx.FxGraphics.drawString(int, int)
未能转换 com.ms.fx.FxGraphics.drawStringFormatted
未能转换 com.ms.fx.FxGraphics.drawStringWithoutFxFont
未能转换 com.ms.fx.FxGraphics.drawT2Curve
未能转换 com.ms.fx.FxGraphics.fill3DRect
未能转换 com.ms.fx.FxGraphics.getClip
未能转换 com.ms.fx.FxGraphics.getExtendedGraphics
未能转换 com.ms.fx.FxGraphics.getGlyphOutline
未能转换 com.ms.fx.FxGraphics.intelliFont
未能转换 com.ms.fx.FxGraphics.setClip
未能转换 com.ms.fx.FxGraphicsOVM.baseGraphics
未能转换 com.ms.fx.FxGraphicsOVM.clearRect
未能转换 com.ms.fx.FxGraphicsOVM.clipRect
未能转换 com.ms.fx.FxGraphicsOVM.copyArea
未能转换 com.ms.fx.FxGraphicsOVM.create
未能转换 com.ms.fx.FxGraphicsOVM.dispose
未能转换 com.ms.fx.FxGraphicsOVM.drawArc
未能转换 com.ms.fx.FxGraphicsOVM.drawBezier
未能转换 com.ms.fx.FxGraphicsOVM.drawBorder
未能转换 com.ms.fx.FxGraphicsOVM.drawBytes
未能转换 com.ms.fx.FxGraphicsOVM.drawChars
未能转换 com.ms.fx.FxGraphicsOVM.drawChars(char[], int, int, int)

未能转换 com.ms.fx.FxGraphicsOVM.drawChars(char[], int, int, int, int, Rectangle, int, int[], int[])
未能转换 com.ms.fx.FxGraphicsOVM.drawCharsWithoutFxFont
未能转换 com.ms.fx.FxGraphicsOVM.drawLine
未能转换 com.ms.fx.FxGraphicsOVM.drawOutlinePolygon
未能转换 com.ms.fx.FxGraphicsOVM.drawOval
未能转换 com.ms.fx.FxGraphicsOVM.drawPixels
未能转换 com.ms.fx.FxGraphicsOVM.drawPolygon
未能转换 com.ms.fx.FxGraphicsOVM.drawPolyline
未能转换 com.ms.fx.FxGraphicsOVM.drawRect
未能转换 com.ms.fx.FxGraphicsOVM.drawRoundRect
未能转换 com.ms.fx.FxGraphicsOVM.drawScanLines
未能转换 com.ms.fx.FxGraphicsOVM.drawString
未能转换 com.ms.fx.FxGraphicsOVM.drawStringWithoutFxFont
未能转换 com.ms.fx.FxGraphicsOVM.excludeClip
未能转换 com.ms.fx.FxGraphicsOVM.fillArc
未能转换 com.ms.fx.FxGraphicsOVM.fillOval
未能转换 com.ms.fx.FxGraphicsOVM.fillPolygon
未能转换 com.ms.fx.FxGraphicsOVM.fillRect
未能转换 com.ms.fx.FxGraphicsOVM.fillRoundRect
未能转换 com.ms.fx.FxGraphicsOVM.FxFxGraphicsOVM
未能转换 com.ms.fx.FxGraphicsOVM.getBaseGraphics
未能转换 com.ms.fx.FxGraphicsOVM.getClipBounds
未能转换 com.ms.fx.FxGraphicsOVM.getClipRect
未能转换 com.ms.fx.FxGraphicsOVM.getClipRegion
未能转换 com.ms.fx.FxGraphicsOVM.getColor
未能转换 com.ms.fx.FxGraphicsOVM.getFont
未能转换 com.ms.fx.FxGraphicsOVM.getFontMetrics
未能转换 com.ms.fx.FxGraphicsOVM.getGlyphOutline
未能转换 com.ms.fx.FxGraphicsOVM.getTranslation
未能转换 com.ms.fx.FxGraphicsOVM.hardClipRect
未能转换 com.ms.fx.FxGraphicsOVM.intersectClip
未能转换 com.ms.fx.FxGraphicsOVM.nativeSetFont
未能转换 com.ms.fx.FxGraphicsOVM.originX
未能转换 com.ms.fx.FxGraphicsOVM.originY
未能转换 com.ms.fx.FxGraphicsOVM.runningAFC
未能转换 com.ms.fx.FxGraphicsOVM.setClip
未能转换 com.ms.fx.FxGraphicsOVM.setClip(int, int, int)
未能转换 com.ms.fx.FxGraphicsOVM.setClip(Region)
未能转换 com.ms.fx.FxGraphicsOVM.setClip(Shape)
未能转换 com.ms.fx.FxGraphicsOVM.setColor

未能转换 com.ms.fx.FxGraphicsOVM.setFont
未能转换 com.ms.fx.FxGraphicsOVM.setPaintMode
未能转换 com.ms.fx.FxGraphicsOVM.setXORMode
未能转换 com.ms.fx.FxGraphicsOVM.systemInterface
未能转换 com.ms.fx.FxGraphicsOVM11
未能转换 com.ms.fx.FxGraphicsOVM11.clearRect
未能转换 com.ms.fx.FxGraphicsOVM11.clipRect
未能转换 com.ms.fx.FxGraphicsOVM11.copyArea
未能转换 com.ms.fx.FxGraphicsOVM11.dispose
未能转换 com.ms.fx.FxGraphicsOVM11.drawArc
未能转换 com.ms.fx.FxGraphicsOVM11.drawBezier
未能转换 com.ms.fx.FxGraphicsOVM11.drawBorder
未能转换 com.ms.fx.FxGraphicsOVM11.drawBytes
未能转换 com.ms.fx.FxGraphicsOVM11.drawChars
未能转换 com.ms.fx.FxGraphicsOVM11.drawCharsWithoutFxFont
未能转换 com.ms.fx.FxGraphicsOVM11.drawLine
未能转换 com.ms.fx.FxGraphicsOVM11.drawOutlinePolygon
未能转换 com.ms.fx.FxGraphicsOVM11.drawOval
未能转换 com.ms.fx.FxGraphicsOVM11.drawPixels
未能转换 com.ms.fx.FxGraphicsOVM11.drawPolygon
未能转换 com.ms.fx.FxGraphicsOVM11.drawPolyline
未能转换 com.ms.fx.FxGraphicsOVM11.drawRect
未能转换 com.ms.fx.FxGraphicsOVM11.drawRoundRect
未能转换 com.ms.fx.FxGraphicsOVM11.drawScanLines
未能转换 com.ms.fx.FxGraphicsOVM11.drawString
未能转换 com.ms.fx.FxGraphicsOVM11.drawStringWithoutFxFont
未能转换 com.ms.fx.FxGraphicsOVM11.excludeClip
未能转换 com.ms.fx.FxGraphicsOVM11.fillArc
未能转换 com.ms.fx.FxGraphicsOVM11.fillOval
未能转换 com.ms.fx.FxGraphicsOVM11.fillPolygon
未能转换 com.ms.fx.FxGraphicsOVM11.fillRect
未能转换 com.ms.fx.FxGraphicsOVM11.fillRoundRect
未能转换 com.ms.fx.FxGraphicsOVM11.getBaseGraphics
未能转换 com.ms.fx.FxGraphicsOVM11.getClipBounds
未能转换 com.ms.fx.FxGraphicsOVM11.getClipRect
未能转换 com.ms.fx.FxGraphicsOVM11.getClipRegion
未能转换 com.ms.fx.FxGraphicsOVM11.getColor
未能转换 com.ms.fx.FxGraphicsOVM11.getFont
未能转换 com.ms.fx.FxGraphicsOVM11.getFontMetrics
未能转换 com.ms.fx.FxGraphicsOVM11.getGlyphOutline

未能转换 com.ms.fx.FxGraphicsOVM11.getTranslation
未能转换 com.ms.fx.FxGraphicsOVM11.intersectClip
未能转换 com.ms.fx.FxGraphicsOVM11.nativeSetFont
未能转换 com.ms.fx.FxGraphicsOVM11.originX
未能转换 com.ms.fx.FxGraphicsOVM11.originY
未能转换 com.ms.fx.FxGraphicsOVM11.runningAFC
未能转换 com.ms.fx.FxGraphicsOVM11.setClip
未能转换 com.ms.fx.FxGraphicsOVM11.setColor
未能转换 com.ms.fx.FxGraphicsOVM11.setFont
未能转换 com.ms.fx.FxGraphicsOVM11.setPaintMode
未能转换 com.ms.fx.FxGraphicsOVM11.setXORMode
未能转换 com.ms.fx.FxGraphicsOVM11.systemInterface
未能转换 com.ms.fx.FxOutlineFont
未能转换 com.ms.fx.FxPen.calcLineValues
未能转换 com.ms.fx.FxPen.drawRoundRectCallback
未能转换 com.ms.fx.FxPen.drawScanLinesCallback
未能转换 com.ms.fx.FxPen.fillRectCallback
未能转换 com.ms.fx.FxPen.FxPen
未能转换 com.ms.fx.FxPen.myDrawOval
未能转换 com.ms.fx.FxRubberPen
未能转换 com.ms.fx.FxStateConfigurableImage
未能转换 com.ms.fx.FxStateConfigurableUIImage.FxStateConfigurableUIImage
未能转换 com.ms.fx.FxStateConfigurableUIImage.getImageState
未能转换 com.ms.fx.FxStyledPen
未能转换 com.ms.fx.FxStyledPen.FxStyledPen
未能转换 com.ms.fx.FxSystemFont
未能转换 com.ms.fx.FxSystemIcon
未能转换 com.ms.fx.FxText.buffer
未能转换 com.ms.fx.FxText.getChar
未能转换 com.ms.fx.FxText.getWordBreak
未能转换 com.ms.fx.FxText.isDelimiter
未能转换 com.ms.fx.FxText.isWhite
未能转换 com.ms.fx.FxText.nChars
未能转换 com.ms.fx.FxText.setText
未能转换 com.ms.fx.FxTexture.DRAW_BL
未能转换 com.ms.fx.FxTexture.DRAW_BOTTOM
未能转换 com.ms.fx.FxTexture.DRAW_BR
未能转换 com.ms.fx.FxTexture.DRAW_CENTER
未能转换 com.ms.fx.FxTexture.DRAW_LEFT
未能转换 com.ms.fx.FxTexture.DRAW_RIGHT

未能转换 com.ms.fx.FxTexture.DRAW_TL
未能转换 com.ms.fx.FxTexture.DRAW_TOP
未能转换 com.ms.fx.FxTexture.DRAW_TR
未能转换 com.ms.fx.FxTexture.drawScanLinesCallback
未能转换 com.ms.fx.FxTexture.FxTexture
未能转换 com.ms.fx.FxTexture.getBottomAxis
未能转换 com.ms.fx.FxTexture.getInnner
未能转换 com.ms.fx.FxTexture.getLeftAxis
未能转换 com.ms.fx.FxTexture.getPinOrigin
未能转换 com.ms.fx.FxTexture.getRightAxis
未能转换 com.ms.fx.FxTexture.getSnapDraw
未能转换 com.ms.fx.FxTexture.getStretch
未能转换 com.ms.fx.FxTexture.getTopAxis
未能转换 com.ms.fx.FxTexture.getUpdatedAreasMask
未能转换 com.ms.fx.FxTexture.imageUpdate
未能转换 com.ms.fx.FxTexture.REPEAT_PIN
未能转换 com.ms.fx.FxTexture.setAxis
未能转换 com.ms.fx.FxTexture.setPinOrigin
未能转换 com.ms.fx.FxTexture.setSnapDraw
未能转换 com.ms.fx.FxTexture.setStretch
未能转换 com.ms.fx.FxTexture.setUpdateCallback
未能转换 com.ms.fx.FxTexture.setUpdatedAreasMask
未能转换 com.ms.fx.FxTexture.size
未能转换 com.ms.fx.FxTexture.SNAP_EDGES
未能转换 com.ms.fx.FxTexture.STRETCH_MIDDLE
未能转换 com.ms.fx.FxTexture.STRETCH_OUTER
未能转换 com.ms.fx.FxToolkit
未能转换 com.ms.fx.GlyphMetrics
未能转换 com.ms.fx.GlyphOutline
未能转换 com.ms.fx.IFxShape
未能转换 com.ms.fx.IFxSystemInterface
未能转换 com.ms.fx.IFxTextCallback
未能转换 com.ms.fx.IFxTextConstants
未能转换 com.ms.fx.IFxTextConstants.DIRAYOUT
未能转换 com.ms.fx.IFxTextConstants.htaCenter
未能转换 com.ms.fx.IFxTextConstants.htaJustified
未能转换 com.ms.fx.IFxTextConstants.htaLeft
未能转换 com.ms.fx.IFxTextConstants.htaRight
未能转换 com.ms.fx.IFxTextConstants.htaScriptDefault
未能转换 com.ms.fx.IFxTextConstants.htaStretch

未能转换 com.ms.fx.IFxTextConstants.MOVE_DOWN
未能转换 com.ms.fx.IFxTextConstants.MOVE_LEFT
未能转换 com.ms.fx.IFxTextConstants.MOVE_RIGHT
未能转换 com.ms.fx.IFxTextConstants.MOVE_UP
未能转换 com.ms.fx.IFxTextConstants.NEXT_DOWN
未能转换 com.ms.fx.IFxTextConstants.NEXT_LEFT
未能转换 com.ms.fx.IFxTextConstants.NEXT_RIGHT
未能转换 com.ms.fx.IFxTextConstants.NEXT_UP
未能转换 com.ms.fx.IFxTextConstants.OPAQUE_BODY
未能转换 com.ms.fx.IFxTextConstants.OPAQUE_POST
未能转换 com.ms.fx.IFxTextConstants.OPAQUE_PRIOR
未能转换 com.ms.fx.IFxTextConstants.SCRIPT_DEF
未能转换 com.ms.fx.IFxTextConstants.tdBt_LR
未能转换 com.ms.fx.IFxTextConstants.tdBt_RL
未能转换 com.ms.fx.IFxTextConstants.tdHebrewNormal
未能转换 com.ms.fx.IFxTextConstants.tdJapanTradNormal
未能转换 com.ms.fx.IFxTextConstants.tdLatinNormal
未能转换 com.ms.fx.IFxTextConstants.tdLeftToRightReading
未能转换 com.ms.fx.IFxTextConstants.tdLR_BT
未能转换 com.ms.fx.IFxTextConstants.tdLR_TB
未能转换 com.ms.fx.IFxTextConstants.tdMongolianNormal
未能转换 com.ms.fx.IFxTextConstants.tdRightToLeftReading
未能转换 com.ms.fx.IFxTextConstants.tdRL_BT
未能转换 com.ms.fx.IFxTextConstants.tdRL_TB
未能转换 com.ms.fx.IFxTextConstants.tdScriptDefault
未能转换 com.ms.fx.IFxTextConstants.tdTB_LR
未能转换 com.ms.fx.IFxTextConstants.tdTB_RL
未能转换 com.ms.fx.IFxTextConstants.tdVisualLayout
未能转换 com.ms.fx.IFxTextConstants.VISUAL_DEF
未能转换 com.ms.fx.IFxTextConstants.vtaBaseline
未能转换 com.ms.fx.IFxTextConstants.vtaBottom
未能转换 com.ms.fx.IFxTextConstants.vtaCenter
未能转换 com.ms.fx.IFxTextConstants.vtaScriptDefault
未能转换 com.ms.fx.IFxTextConstants.vtaStretch
未能转换 com.ms.fx.IFxTextConstants.vtaTop
未能转换 com.ms.fx.IFxTextConstants.wwCleanEdges
未能转换 com.ms.fx.IFxTextConstants.wwKeepWordIntact
未能转换 com.ms.fx.IFxTextConstants.wwMask
未能转换 com.ms.fx.IFxTextConstants.wwNone
未能转换 com.ms.fx.IFxTextConstants.wwTypeMask

未能转换 com.ms.fx.IFxTextConstants.wwVirtualRectEnd
未能转换 com.ms.fx.IFxTextConstants.wwVirtualRectSide
未能转换 com.ms.fx.IFxTextConstants.wwWrap
未能转换 com.ms.fx.IFxTextureUpdate
未能转换 com.ms.fx.OutlineCurve
未能转换 com.ms.fx.OutlinePolygon
未能转换 com.ms.fx.PeerConstants
未能转换 com.ms.fx.Region.clone
未能转换 com.ms.fx.Region.COMPLEX
未能转换 com.ms.fx.Region.complexity
未能转换 com.ms.fx.Region.copy
未能转换 com.ms.fx.Region.EMPTY~
未能转换 com.ms.fx.Region.equals
未能转换 com.ms.fx.Region.fillBounds
未能转换 com.ms.fx.Region.getBounds
未能转换 com.ms.fx.Region.getGeometry
未能转换 com.ms.fx.Region.invert
未能转换 com.ms.fx.Region.isEmpty
未能转换 com.ms.fx.Region.Region
未能转换 com.ms.fx.Region.set
未能转换 com.ms.fx.Region.SIMPLE
未能转换 com.ms.fx.RegionConverter
未能转换 com.ms.fx.txtRun
未能转换 com.ms.fx.Version

Com.ms.io 错误信息

未能转换 com.ms.io.console.Console
未能转换 com.ms.io.console.DefaultConsole
未能转换 com.ms.io.ObjectInputStreamWithLoader
未能转换 com.ms.io.OffsetInputStreamFilter.mark
未能转换 com.ms.io.OffsetInputStreamFilter.reset
未能转换 com.ms.io.Path.hasExecExtensionType
未能转换 com.ms.io.Path.isRoot
未能转换 com.ms.io.Path.validateFilename
未能转换 com.ms.io.SystemInputStream
未能转换 com.ms.io.SystemOutputStream
未能转换 com.ms.io.UserFileDialog

Com.ms.jdbc.odbc 错误信息

未能转换 com.ms.jdbc.odbc.JdbcOdbcConnection.getAutoCommit

未能转换 com.ms.jdbc.odbc.JdbcOdbcConnection.getMetaData

Com.ms.lang 错误信息

未能转换 com.ms.lang.MulticastDelegate.invokeHelperMulticast
未能转换 com.ms.lang.RegKey.enumKey
未能转换 com.ms.lang.RegKey.finalize
未能转换 com.ms.lang.RegKey.getBinaryValue
未能转换 com.ms.lang.RegKey.KEYOPEN_ALL
未能转换 com.ms.lang.RegKey.KEYOPEN_CREATE
未能转换 com.ms.lang.RegKey.KEYOPEN_READ
未能转换 com.ms.lang.RegKey.KEYOPEN_WRITE
未能转换 com.ms.lang.RegKey.loadKey
未能转换 com.ms.lang.RegKey.queryInfo
未能转换 com.ms.lang.RegKey.replace
未能转换 com.ms.lang.RegKey.restore
未能转换 com.ms.lang.RegKey.unload
未能转换 com.ms.lang.RegQueryInfo
未能转换 com.ms.lang.SystemThread
未能转换 com.ms.lang.SystemX.arrayCompare
未能转换 com.ms.lang.SystemX.blockcopy
未能转换 com.ms.lang.SystemX.exitProcessAfterMainThreadReturns
未能转换 com.ms.lang.SystemX.getDeclaredMethodFromSignature
未能转换 com.ms.lang.SystemX.getDefaultInputManager
未能转换 com.ms.lang.SystemX.getInputManager
未能转换 com.ms.lang.SystemX.getKeyboardLanguageName
未能转换 com.ms.lang.SystemX.getKeyboardLanguages
未能转换 com.ms.lang.SystemX.getMethodFromSignature
未能转换 com.ms.lang.SystemX.getNativeServices
未能转换 com.ms.lang.SystemX.isLocalCharDBCSLeadByte
未能转换 com.ms.lang.SystemX.JavaStringToLocalString
未能转换 com.ms.lang.SystemX.LocalStringToJavaString
未能转换 com.ms.lang.SystemX.setInputManager
未能转换 com.ms.lang.SystemX.setKeyboardLanguage
未能转换 com.ms.lang.VerifyErrorEx

Com.ms.mtx 错误信息

未能转换 com.ms.mtx.AppServer
未能转换 com.ms.mtx.Context.createObject
未能转换 com.ms.mtx.Context.disableCommit
未能转换 com.ms.mtx.Context.enableCommit
未能转换 com.ms.mtx.Context.getContextId
未能转换 com.ms.mtx.Context.getDeactivateOnReturn
未能转换 com.ms.mtx.Context.getDirectCallerName
未能转换 com.ms.mtx.Context.getDirectCreatorName
未能转换 com.ms.mtx.Context.getMyTransactionVote
未能转换 com.ms.mtx.Context.getObjectContext
未能转换 com.ms.mtx.Context.getOriginalCallerName
未能转换 com.ms.mtx.Context.getOriginalCreatorName
未能转换 com.ms.mtx.Context.getProperty
未能转换 com.ms.mtx.Context.getPropertyNames
未能转换 com.ms.mtx.Context.getSafeRef
未能转换 com.ms.mtx.Context.getTransaction
未能转换 com.ms.mtx.Context.getTransactionId
未能转换 com.ms.mtx.Context.isCallerInRole
未能转换 com.ms.mtx.Context.isInTransaction
未能转换 com.ms.mtx.Context.isSecurityEnabled
未能转换 com.ms.mtx.Context.setAbort
未能转换 com.ms.mtx.Context.setComplete
未能转换 com.ms.mtx.Context.setDeactivateOnReturn
未能转换 com.ms.mtx.Context.setMyTransactionVote
未能转换 com.ms.mtx.Context.TxAbsort
未能转换 com.ms.mtx.Context.TxCommit
未能转换 com.ms.mtx.IContextState.GetDeactivateOnReturn
未能转换 com.ms.mtx.IContextState.GetMyTransactionVote
未能转换 com.ms.mtx.IContextState.iid
未能转换 com.ms.mtx.IContextState.SetDeactivateOnReturn
未能转换 com.ms.mtx.IContextState.SetMyTransactionVote
未能转换 com.ms.mtx.IEnumNames
未能转换 com.ms.mtx.IGetContextProperties
未能转换 com.ms.mtx.IMTxAS
未能转换 com.ms.mtx.IMITxAS.GetObjectContext
未能转换 com.ms.mtx.IMITxAS.iid
未能转换 com.ms.mtx.IObjectContext.CreateInstance

未能转换 com.ms.mtx.IObjectContext.iid
未能转换 com.ms.mtx.IObjectContextInfo.iid
未能转换 com.ms.mtx.IObjectControl.iid
未能转换 com.ms.mtx.IObjectControl
未能转换 com.ms.mtx.ISecurityCallContext.getItem
未能转换 com.ms.mtx.ISecurityCallContext.iid
未能转换 com.ms.mtx.ISecurityCallersColl.iid
未能转换 com.ms.mtx.ISecurityIdentityColl.getItem
未能转换 com.ms.mtx.ISecurityIdentityColl.iid
未能转换 com.ms.mtx.SecurityProperty.GetDirectCallerName
未能转换 com.ms.mtx.SecurityProperty.GetOriginalCallerName
未能转换 com.ms.mtx.ISharedProperty.iid
未能转换 com.ms.mtx.ISharedPropertyGroup.iid
未能转换 com.ms.mtx.ISharedPropertyGroupManager.iid
未能转换 com.ms.mtx.ITransactionContextEx
未能转换 com.ms.mtx.MTx
未能转换 com.ms.mtx.ObjectContext.CreateInstance
未能转换 com.ms.mtx.ObjectContext.DisableCommit
未能转换 com.ms.mtx.ObjectContext.EnableCommit
未能转换 com.ms.mtx.ObjectContext.get_NewEnum
未能转换 com.ms.mtx.ObjectContext.getCount
未能转换 com.ms.mtx.ObjectContext.getItem
未能转换 com.ms.mtx.ObjectContext.getSecurity
未能转换 com.ms.mtx.ObjectContext.iid
未能转换 com.ms.mtx.ObjectContext.IsCallerInRole
未能转换 com.ms.mtx.ObjectContext.IsInTransaction
未能转换 com.ms.mtx.ObjectContext.IsSecurityEnabled
未能转换 com.ms.mtx.ObjectContext.SetAbort
未能转换 com.ms.mtx.ObjectContext.SetComplete
未能转换 com.ms.mtx.ISecurityCallContext.getCallers
未能转换 com.ms.mtx.ISecurityCallContext.getDirectCaller
未能转换 com.ms.mtx.ISecurityCallContext.getNumCallers
未能转换 com.ms.mtx.ISecurityCallContext.getProperty
未能转换 com.ms.mtx.ISecurityCallContext.getPropertyNames
未能转换 com.ms.mtx.ISecurityCallContext.isUserInRole
未能转换 com.ms.mtx.ISecurityCaller
未能转换 com.ms.mtx.SecurityProperty.GetDirectCreatorName
未能转换 com.ms.mtx.SecurityProperty.GetOriginalCreatorName
未能转换 com.ms.mtx.SecurityProperty.iid
未能转换 com.ms.mtx.SharedPropertyGroupManager.clsid

未能转换 com.ms.mtx.SharedPropertyGroupManager.get_NewEnum

未能转换 com.ms.mtx.TransactionContextEx

Com.ms.object 错误信息

未能转换 com.ms.object.Category
未能转换 com.ms.object.dragdrop.DragHandler
未能转换 com.ms.object.dragdrop.DragHelper
未能转换 com.ms.object.dragdrop.DragProxy
未能转换 com.ms.object.dragdrop.DragSession.getDragModifiers
未能转换 com.ms.object.dragdrop.DragSource.DEFAULT_ACTION
未能转换 com.ms.object.dragdrop.DragSource.queryDragCursor
未能转换 com.ms.object.dragdrop.DragSource.queryDragStatus
未能转换 com.ms.object(IServiceObjectProvider
未能转换 com.ms.object.ISite
未能转换 com.ms.object.ISiteable
未能转换 com.ms.object.MetaObject
未能转换 com.ms.object.ObjectBag
未能转换 com.ms.object.SimpleTransferSession
未能转换 com.ms.object.TransferSession

Com.ms.ui 错误信息

未能转换 com.ms.ui.<ClassName>.add*Listener
未能转换 com.ms.ui.<ClassName>.addNotify
未能转换 com.ms.ui.<ClassName>.getPeer
未能转换 com.ms.ui.<ClassName>.handleEvent
未能转换 com.ms.ui.<ClassName>.isNotified
未能转换 com.ms.ui.<ClassName>.process*Event
未能转换 com.ms.ui.<ClassName>.remove*Listener
未能转换 com.ms.ui.<ClassName>.removeNotify
未能转换 com.ms.ui.<ClassName>.setListenerTracker
未能转换 com.ms.ui.<ClassName> BeanInfo
未能转换 com.ms.ui.AwtUIApplet.add(IUIComponent)
未能转换 com.ms.ui.AwtUIApplet.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUIApplet.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIApplet.add(IUIComponent, Object, int)
未能转换 com.ms.ui.AwtUIApplet.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIApplet.AwtUIApplet(IUIComponent)
未能转换 com.ms.ui.AwtUIApplet.AwtUIApplet(UIApplet)
未能转换 com.ms.ui.AwtUIApplet.destroy
未能转换 com.ms.ui.AwtUIApplet.getComponent
未能转换 com.ms.ui.AwtUIApplet.getHeader
未能转换 com.ms.ui.AwtUIApplet.getRoot
未能转换 com.ms.ui.AwtUIApplet.getTaskManager
未能转换 com.ms.ui.AwtUIApplet.getUIComponentCount
未能转换 com.ms.ui.AwtUIApplet.setHeader
未能转换 com.ms.ui.AwtUIApplet.setLayout
未能转换 com.ms.ui.AwtUIBand
未能转换 com.ms.ui.AwtUIBandBeanInfo.AwtUIBandBeanInfo
未能转换 com.ms.ui.AwtUIBandBeanInfo.getIcon
未能转换 com.ms.ui.AwtUIBandBox
未能转换 com.ms.ui.AwtUIBandBoxBeanInfo.AwtUIBandBoxBeanInfo
未能转换 com.ms.ui.AwtUIBandBoxBeanInfo.getIcon
未能转换 com.ms.ui.AwtUIButton._btn
未能转换 com.ms.ui.AwtUIButton.getStyle
未能转换 com.ms.ui.AwtUIButton.keyDown
未能转换 com.ms.ui.AwtUIButton.keyUp
未能转换 com.ms.ui.AwtUIButton.mouseUp
未能转换 com.ms.ui.AwtUIButton.setStyle

未能转换 com.ms.ui.AwtUICheckButton.getBase
未能转换 com.ms.ui.AwtUICheckButton.getSelectedObjects
未能转换 com.ms.ui.AwtUIChoice.add
未能转换 com.ms.ui.AwtUIChoice.addSelectedIndex
未能转换 com.ms.ui.AwtUIChoice.addSelectedIndices
未能转换 com.ms.ui.AwtUIChoice.addSelectedItem
未能转换 com.ms.ui.AwtUIChoice.addSelectedItems
未能转换 com.ms.ui.AwtUIChoice.getAnchorItem
未能转换 com.ms.ui.AwtUIChoice.getBase
未能转换 com.ms.ui.AwtUIChoicegetExtensionItem
未能转换 com.ms.ui.AwtUIChoice.getSelectedItem
未能转换 com.ms.ui.AwtUIChoice.getSelectedItems
未能转换 com.ms.ui.AwtUIChoice.getSelectedObjects
未能转换 com.ms.ui.AwtUIChoice.getSelectionMode
未能转换 com.ms.ui.AwtUIChoice.getStyle
未能转换 com.ms.ui.AwtUIChoice.removeSelectedIndex
未能转换 com.ms.ui.AwtUIChoice.removeSelectedIndices
未能转换 com.ms.ui.AwtUIChoice.removeSelectedItem
未能转换 com.ms.ui.AwtUIChoice.removeSelectedItems
未能转换 com.ms.ui.AwtUIChoice.setAnchorItem
未能转换 com.ms.ui.AwtUIChoice.setExtensionItem
未能转换 com.ms.ui.AwtUIChoice.setSelectedIndex(int)
未能转换 com.ms.ui.AwtUIChoice.setSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUIChoice.setSelectedIndices
未能转换 com.ms.ui.AwtUIChoice.setSelectedItem
未能转换 com.ms.ui.AwtUIChoice.setSelectedItems
未能转换 com.ms.ui.AwtUIChoice.setSelectionMode
未能转换 com.ms.ui.AwtUIChoice.setStyle
未能转换 com.ms.ui.AwtUIColumnViewer
未能转换 com.ms.ui.AwtUIControl.add(Component, Object)
未能转换 com.ms.ui.AwtUIControl.add(Component, Object, int)
未能转换 com.ms.ui.AwtUIControl.add(IUIComponent)
未能转换 com.ms.ui.AwtUIControl.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUIControl.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIControl.add(IUIComponent, Object, int)
未能转换 com.ms.ui.AwtUIControl.add(String, Component)
未能转换 com.ms.ui.AwtUIControl.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIControl.AwtUIControl
未能转换 com.ms.ui.AwtUIControl.getID
未能转换 com.ms.ui.AwtUIControl.isSelected

未能转换 com.ms.ui.AwtUIControl.postEvent
未能转换 com.ms.ui.AwtUIControl.setChecked
未能转换 com.ms.ui.AwtUIControl.setHot
未能转换 com.ms.ui.AwtUIControl.setId
未能转换 com.ms.ui.AwtUIControl.setIndeterminate
未能转换 com.ms.ui.AwtUIControl.setLayout
未能转换 com.ms.ui.AwtUIControl.setPressed
未能转换 com.ms.ui.AwtUIControl.setReparent
未能转换 com.ms.ui.AwtUIControl.setSelected
未能转换 com.ms.ui.AwtUIDialog.add(IUIComponent)
未能转换 com.ms.ui.AwtUIDialog.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUIDialog.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIDialog.add(IUIComponent, Object, int)
未能转换 com.ms.ui.AwtUIDialog.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIDialog.getComponent
未能转换 com.ms.ui.AwtUIDialog.getPosition
未能转换 com.ms.ui.AwtUIDialog.setLayout
未能转换 com.ms.ui.AwtUIDrawText.getCharFromScreen
未能转换 com.ms.ui.AwtUIDrawText.getCharLocation
未能转换 com.ms.ui.AwtUIDrawText.getOutline
未能转换 com.ms.ui.AwtUIDrawText.isAutoResizable
未能转换 com.ms.ui.AwtUIDrawText.setAutoResizable
未能转换 com.ms.ui.AwtUIDrawText.setHorizAlign
未能转换 com.ms.ui.AwtUIDrawText.setOutline
未能转换 com.ms.ui.AwtUIDrawText.setRefresh
未能转换 com.ms.ui.AwtUIDrawText.setVertAlign
未能转换 com.ms.ui.AwtUIDrawText.setWordWrap
未能转换 com.ms.ui.AwtUIEdit.getCharFromScreen
未能转换 com.ms.ui.AwtUIEdit.getCharLocation
未能转换 com.ms.ui.AwtUIEdit.getOutline
未能转换 com.ms.ui.AwtUIEdit.getWordWrap
未能转换 com.ms.ui.AwtUIEdit.isAutoResizable
未能转换 com.ms.ui.AwtUIEdit.setAutoResizable
未能转换 com.ms.ui.AwtUIEdit.setHorizAlign
未能转换 com.ms.ui.AwtUIEdit.setOutline
未能转换 com.ms.ui.AwtUIEdit.setRefresh
未能转换 com.ms.ui.AwtUIEdit.setVertAlign
未能转换 com.ms.ui.AwtUIEdit.setWordWrap
未能转换 com.ms.ui.AwtUIEdit.showCaret
未能转换 com.ms.ui.AwtUIFrame.add(IUIComponent)

未能转换 com.ms.ui.AwtUIFrame.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUIFrame.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIFrame.add(IUIComponent, Object, int)
未能转换 com.ms.ui.AwtUIFrame.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIFrame.requestFocus
未能转换 com.ms.ui.AwtUIFrame.setLayout
未能转换 com.ms.ui.AwtUIGraphic.AwtUIGraphic
未能转换 com.ms.ui.AwtUIGraphic.getContentBounds
未能转换 com.ms.ui.AwtUIGraphic.imageUpdate
未能转换 com.ms.ui.AwtUIHost
未能转换 com.ms.ui.AwtUIHost.add(IUIComponent)
未能转换 com.ms.ui.AwtUIHost.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUIHost.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIHost.add(IUIComonent, Object, int)
未能转换 com.ms.ui.AwtUIHost.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIHost.disableHostEvents
未能转换 com.ms.ui.AwtUIHost.enableHostEvents
未能转换 com.ms.ui.AwtUIHost.getComponent
未能转换 com.ms.ui.AwtUIHost.getHeader
未能转换 com.ms.ui.AwtUIHost.getPreferredSize
未能转换 com.ms.ui.AwtUIHost.getRoot
未能转换 com.ms.ui.AwtUIHost.getUIComponent
未能转换 com.ms.ui.AwtUIHost.invalidate
未能转换 com.ms.ui.AwtUIHost.layout
未能转换 com.ms.ui.AwtUIHost.listenerTracker
未能转换 com.ms.ui.AwtUIHost.obtainListenerTracker
未能转换 com.ms.ui.AwtUIHost.paint
未能转换 com.ms.ui.AwtUIHost.paintAll
未能转换 com.ms.ui.AwtUIHost.preferredSize
未能转换 com.ms.ui.AwtUIHost.preProcessHostEvent
未能转换 com.ms.ui.AwtUIHost.root
未能转换 com.ms.ui.AwtUIHost.setHeader
未能转换 com.ms.ui.AwtUIHost.setLayout
未能转换 com.ms.ui.AwtUIHost.setListenerHost
未能转换 com.ms.ui.AwtUIHost.show
未能转换 com.ms.ui.AwtUIHost.usingNewEvents
未能转换 com.ms.ui.AwtUIHost.validate
未能转换 com.ms.ui.AwtUIHost.validateTree
未能转换 com.ms.ui.AwtUIList.addSelectedIndex(int)
未能转换 com.ms.ui.AwtUIList.addSelectedIndex(int, boolean)

未能转换 com.ms.ui.AwtUIList.addSelectedIndices(int[])
未能转换 com.ms.ui.AwtUIList.addSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUIList.addSelectedItem
未能转换 com.ms.ui.AwtUIList.addSelectedItems
未能转换 com.ms.ui.AwtUIList.AwtUIList(int)
未能转换 com.ms.ui.AwtUIList.AwtUIList(int, int)
未能转换 com.ms.ui.AwtUIList.find
未能转换 com.ms.ui.AwtUIList.getSelectedIndex
未能转换 com.ms.ui.AwtUIList.getSelectedItem
未能转换 com.ms.ui.AwtUIList.remove
未能转换 com.ms.ui.AwtUIList.removeSelectedIndex(int)
未能转换 com.ms.ui.AwtUIList.removeSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUIList.removeSelectedIndices(int[])
未能转换 com.ms.ui.AwtUIList.removeSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUIList.removeSelectedItem
未能转换 com.ms.ui.AwtUIList.removeSelectedItems
未能转换 com.ms.ui.AwtUIList.setSelectedIndex(int)
未能转换 com.ms.ui.AwtUIList.setSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUIList.setSelectedIndices(int[])
未能转换 com.ms.ui.AwtUIList.setSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUIList.setSelectedItem
未能转换 com.ms.ui.AwtUIList.setSelectedItems
未能转换 com.ms.ui.AwtUIList.setSelectionMode
未能转换 com.ms.ui.AwtUIMarquee
未能转换 com.ms.ui.AwtUIMenuList.AwtUIMenuList
未能转换 com.ms.ui.AwtUIMenuList.getSelectedObjects
未能转换 com.ms.ui.AwtUIMessageBox
未能转换 com.ms.ui.AwtUIMessageBox.action
未能转换 com.ms.ui.AwtUIMessageBox.AwtUIMessageBox
未能转换 com.ms.ui.AwtUIMessageBox.doModal
未能转换 com.ms.ui.AwtUIMessageBox.doModalIndex
未能转换 com.ms.ui.AwtUIMessageBox.getButtonAlignment
未能转换 com.ms.ui.AwtUIMessageBox.getButtons
未能转换 com.ms.ui.AwtUIMessageBox.getDefaultButton
未能转换 com.ms.ui.AwtUIMessageBox.getFrame
未能转换 com.ms.ui.AwtUIMessageBox.getImage
未能转换 com.ms.ui.AwtUIMessageBox.getText
未能转换 com.ms.ui.AwtUIMessageBox.getTimeout
未能转换 com.ms.ui.AwtUIMessageBox Insets
未能转换 com.ms.ui.AwtUIMessageBox.keyDown

未能转换 com.ms.ui.AwtUIMessageBox.keyUp
未能转换 com.ms.ui.AwtUIMessageBox.preferredSize
未能转换 com.ms.ui.AwtUIMessageBox.setButtonAlignment
未能转换 com.ms.ui.AwtUIMessageBox.setButtons
未能转换 com.ms.ui.AwtUIMessageBox.setDefaultButton
未能转换 com.ms.ui.AwtUIMessageBox.setImage
未能转换 com.ms.ui.AwtUIMessageBox.setText
未能转换 com.ms.ui.AwtUIMessageBox.setTimeout
未能转换 com.ms.ui.AwtUIMessageBox.timeTriggered
未能转换 com.ms.ui.AwtUIPanel.setLayout
未能转换 com.ms.ui.AwtUIProgress.AwtUIProgress
未能转换 com.ms.ui.AwtUIProgress.getBase
未能转换 com.ms.ui.AwtUIPushButton.getBase
未能转换 com.ms.ui.AwtUIRadioButton.AwtUIRadioButton(IUIComponent, int)
未能转换 com.ms.ui.AwtUIRadioButton.AwtUIRadioButton(String, int)
未能转换 com.ms.ui.AwtUIRadioButton.getBase
未能转换 com.ms.ui.AwtUIRadioButton.getSelectedObjects
未能转换 com.ms.ui.AwtUIRepeatButton.AwtUIRepeatButton
未能转换 com.ms.ui.AwtUIScrollBar
未能转换 com.ms.ui.AwtUIScrollBar.AwtUIScrollBar
未能转换 com.ms.ui.AwtUIScrollBar.getBase
未能转换 com.ms.ui.AwtUIScrollBar.getStyle
未能转换 com.ms.ui.AwtUIScrollBar.scrollLineDown
未能转换 com.ms.ui.AwtUIScrollBar.scrollLineUp
未能转换 com.ms.ui.AwtUIScrollBar.scrollPageDown
未能转换 com.ms.ui.AwtUIScrollBar.scrollPageUp
未能转换 com.ms.ui.AwtUIScrollBar.setScrollInfo
未能转换 com.ms.ui.AwtUIScrollBar.setScrollLine
未能转换 com.ms.ui.AwtUIScrollBar.setStyle
未能转换 com.ms.ui.AwtUIScrollBar.setUnitIncrement
未能转换 com.ms.ui.AwtUIScrollView
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(Component)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(int, int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(int, int, int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent, int, int)

未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent, int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent, int, int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.AwtUIScrollView(IUIComponent, int, int, int, int, int)
未能转换 com.ms.ui.AwtUIScrollView.getContent
未能转换 com.ms.ui.AwtUIScrollView.getLine
未能转换 com.ms.ui.AwtUIScrollView.getVLine
未能转换 com.ms.ui.AwtUIScrollView.setContent
未能转换 com.ms.ui.AwtUIScrollView.setHLine
未能转换 com.ms.ui.AwtUIScrollView.setLine
未能转换 com.ms.ui.AwtUIScrollView.setVLine
未能转换 com.ms.ui.AwtUISplitViewer
未能转换 com.ms.ui.AwtUISplitViewer.add
未能转换 com.ms.ui.AwtUISplitViewer.AwtUISplitViewer
未能转换 com.ms.ui.AwtUISplitViewer.getComponent
未能转换 com.ms.ui.AwtUISplitViewer.getPos
未能转换 com.ms.ui.AwtUISplitViewer.getStyle
未能转换 com.ms.ui.AwtUISplitViewer.remove
未能转换 com.ms.ui.AwtUISplitViewer.setPos
未能转换 com.ms.ui.AwtUIStatus.getBase
未能转换 com.ms.ui.AwtUITabList
未能转换 com.ms.ui.AwtUITabList.add(IUIComponent)
未能转换 com.ms.ui.AwtUITabList.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUITabList.add(String)
未能转换 com.ms.ui.AwtUITabViewer.add(String, Component)
未能转换 com.ms.ui.AwtUITabViewer.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUITabViewer.addTab
未能转换 com.ms.ui.AwtUITree
未能转换 com.ms.ui.AwtUITree.add(Component)
未能转换 com.ms.ui.AwtUITree.add(Component, int)
未能转换 com.ms.ui.AwtUITree.add(Image, String, int)
未能转换 com.ms.ui.AwtUITree.add(IUIComponent)
未能转换 com.ms.ui.AwtUITree.add(IUIComponent, int)
未能转换 com.ms.ui.AwtUITree.add(String, int)
未能转换 com.ms.ui.AwtUITree.addSelectedIndex(int)
未能转换 com.ms.ui.AwtUITree.addSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUITree.addSelectedIndices(int[])
未能转换 com.ms.ui.AwtUITree.addSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUITree.addSelectedItem(IUIComponent)
未能转换 com.ms.ui.AwtUITree.addSelectedItem(IUIComponent, boolean)

未能转换 com.ms.ui.AwtUITree.addSelectedItems(IUIComponent[])
未能转换 com.ms.ui.AwtUITree.addSelectedItems(IUIComponent[], boolean)
未能转换 com.ms.ui.AwtUITree.getExpander
未能转换 com.ms.ui.AwtUITree.getSelectedIndex
未能转换 com.ms.ui.AwtUITree.getSelectedIndices
未能转换 com.ms.ui.AwtUITree.getSelectedItems
未能转换 com.ms.ui.AwtUITree.getSelectedObjects
未能转换 com.ms.ui.AwtUITree.getSelectionMode
未能转换 com.ms.ui.AwtUITree.remove
未能转换 com.ms.ui.AwtUITree.removeSelectedIndex(int)
未能转换 com.ms.ui.AwtUITree.removeSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUITree.removeSelectedIndices(int[])
未能转换 com.ms.ui.AwtUITree.removeSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUITree.removeSelectedItem(IUIComponent)
未能转换 com.ms.ui.AwtUITree.removeSelectedItem(IUIComponent, boolean)
未能转换 com.ms.ui.AwtUITree.removeSelectedItems(IUIComponent[])
未能转换 com.ms.ui.AwtUITree.removeSelectedItems(IUIComponent[], boolean)
未能转换 com.ms.ui.AwtUITree.setSelectedIndex(int)
未能转换 com.ms.ui.AwtUITree.setSelectedIndex(int, boolean)
未能转换 com.ms.ui.AwtUITree.setSelectedIndices(int[])
未能转换 com.ms.ui.AwtUITree.setSelectedIndices(int[], boolean)
未能转换 com.ms.ui.AwtUITree.setSelectedItem
未能转换 com.ms.ui.AwtUITree.setSelectedItems(IUIComponent[])
未能转换 com.ms.ui.AwtUITree.setSelectedItems(IUIComponent[], boolean)
未能转换 com.ms.ui.AwtUITree.setSelectionMode
未能转换 com.ms.ui.AwtUIWindow.add(IUIComponent)
未能转换 com.ms.ui.AwtUIWindow.add(IUIComponent, Object)
未能转换 com.ms.ui.AwtUIWindow.add(IUIComponent, Object, int)
未能转换 com.ms.ui.AwtUIWindow.add(String, IUIComponent)
未能转换 com.ms.ui.AwtUIWindow.AwtUIWindow
未能转换 com.ms.ui.AwtUIWindow.requestFocus
未能转换 com.ms.ui.AwtUIWindow.setLayout
未能转换 com.ms.ui.ButtonFlowLayout
未能转换 com.ms.ui.ButtonFlowLayout.ButtonFlowLayout
未能转换 com.ms.ui.ButtonFlowLayout.computeUnitDimension
未能转换 com.ms.ui.ButtonFlowLayout.getHeightPad
未能转换 com.ms.ui.ButtonFlowLayout.getMinHeight
未能转换 com.ms.ui.ButtonFlowLayout.getMinWidth
未能转换 com.ms.ui.ButtonFlowLayout.getWidthPad
未能转换 com.ms.ui.ButtonFlowLayout.setHeightPad

未能转换 com.ms.ui.ButtonFlowLayout.setHeight
未能转换 com.ms.ui.ButtonFlowLayout.setWidth
未能转换 com.ms.ui.ButtonFlowLayout.setWidthPad
未能转换 com.ms.ui.ButtonPanel.ButtonPanel
未能转换 com.ms.ui.ButtonPanel.getDefaultButton
未能转换 com.ms.ui.event.UIActionEvent.<EventType>
未能转换 com.ms.ui.event.UIActionEvent.getActionCommand
未能转换 com.ms.ui.event.UIAdjustmentEvent.<EventType>
未能转换 com.ms.ui.event.UIBaseEvent.getId
未能转换 com.ms.ui.event.UIContainerEvent.<EventType>
未能转换 com.ms.ui.event.UIEvent
未能转换 com.ms.ui.event.UIFocusEvent.<EventType>
未能转换 com.ms.ui.event.UIFocusEvent.getArg
未能转换 com.ms.ui.event.UIFocusEvent.isTemporary
未能转换 com.ms.ui.event.UILInputEvent
未能转换 com.ms.ui.event.UILItemEvent.<EventType>
未能转换 com.ms.ui.event.UILItemEvent.getItem
未能转换 com.ms.ui.event.UILItemEvent.getStateChange
未能转换 com.ms.ui.event.UIKeyEvent.CHAR_UNDEFINED
未能转换 com.ms.ui.event.UIKeyEvent.getKeyChar
未能转换 com.ms.ui.event.UIKeyEvent.getKeyCode
未能转换 com.ms.ui.event.UIKeyEvent.getOldEventKey
未能转换 com.ms.ui.event.UIKeyEvent.KEY_EVENT_BASE
未能转换 com.ms.ui.event.UIKeyEvent.KEY_PRESSED
未能转换 com.ms.ui.event.UIKeyEvent.KEY_RELEASED
未能转换 com.ms.ui.event.UIKeyEvent.KEY_TYPED
未能转换 com.ms.ui.event.UIKeyEvent.VK_BACK_QUOTE
未能转换 com.ms.ui.event.UIKeyEvent.VK_EQUALS
未能转换 com.ms.ui.event.UIKeyEvent.VK_META
未能转换 com.ms.ui.event.UIKeyEvent.VK_UNDEFINED
未能转换 com.ms.ui.event.UIMouseEvent.<EventType>
未能转换 com.ms.ui.event.UIMouseEvent.getClickCount
未能转换 com.ms.ui.event.UIMouseEvent.getPoint
未能转换 com.ms.ui.event.UIMouseEvent.getX
未能转换 com.ms.ui.event.UIMouseEvent.getY
未能转换 com.ms.ui.event.UIMouseEvent.isPopupTrigger
未能转换 com.ms.ui.event.UINotifyEvent
未能转换 com.ms.ui.event.UITextEvent.<EventType>
未能转换 com.ms.ui.event.UIWindowEvent.<EventType>
未能转换 com.ms.ui.IAwtUIAdjustable

未能转换 com.ms.ui.IAwtUIItemSelectable
未能转换 com.ms.ui.IUIAccessible
未能转换 com.ms.ui.IUIAccessible.<ErrorCode>
未能转换 com.ms.ui.IUIAccessible.getBounds
未能转换 com.ms.ui.IUIAccessible.navigate
未能转换 com.ms.ui.IUIBand
未能转换 com.ms.ui.IUIComponent.action
未能转换 com.ms.ui.IUIComponent.adjustLayoutSize
未能转换 com.ms.ui.IUIComponent.deliverEvent
未能转换 com.ms.ui.IUIComponent.ensureVisible
未能转换 com.ms.ui.IUIComponent.getBounds
未能转换 com.ms.ui.IUIComponent.getPreferredSize
未能转换 com.ms.ui.IUIComponent.getID
未能转换 com.ms.ui.IUIComponent.getLocation
未能转换 com.ms.ui.IUIComponent.getMaximumSize
未能转换 com.ms.ui.IUIComponent.getMinimumSize
未能转换 com.ms.ui.IUIComponent.getPreferredSize
未能转换 com.ms.ui.IUIComponent.getToolkit
未能转换 com.ms.ui.IUIComponent.isChecked
未能转换 com.ms.ui.IUIComponent.isHeightRelative
未能转换 com.ms.ui.IUIComponent.isHot
未能转换 com.ms.ui.IUIComponent.isIndeterminate
未能转换 com.ms.ui.IUIComponent.isInvalidating
未能转换 com.ms.ui.IUIComponent.isKeyable
未能转换 com.ms.ui.IUIComponent.isKeyable(boolean)
未能转换 com.ms.ui.IUIComponent.isPressed
未能转换 com.ms.ui.IUIComponent.isRedrawing
未能转换 com.ms.ui.IUIComponent.isSelected
未能转换 com.ms.ui.IUIComponent.isValid
未能转换 com.ms.ui.IUIComponent.isWidthRelative
未能转换 com.ms.ui.IUIComponent.keyDown
未能转换 com.ms.ui.IUIComponent.keyUp
未能转换 com.ms.ui.IUIComponent.lostFocus
未能转换 com.ms.ui.IUIComponent.mouseClicked
未能转换 com.ms.ui.IUIComponent.mouseDown
未能转换 com.ms.ui.IUIComponent.mouseEnter
未能转换 com.ms.ui.IUIComponent.mouseExit
未能转换 com.ms.ui.IUIComponent.mouseMove
未能转换 com.ms.ui.IUIComponent.mouseUp
未能转换 com.ms.ui.IUIComponent.paint

未能转换 com.ms.ui.IUIComponent.paintAll
未能转换 com.ms.ui.IUIComponent.prepareImage
未能转换 com.ms.ui.IUIComponent.print
未能转换 com.ms.ui.IUIComponent.printAll
未能转换 com.ms.ui.IUIComponent.recalcPreferredSize
未能转换 com.ms.ui.IUIComponent.setBounds
未能转换 com.ms.ui.IUIComponent.setChecked
未能转换 com.ms.ui.IUIComponent.setFlags
未能转换 com.ms.ui.IUIComponent.setHot
未能转换 com.ms.ui.IUIComponent.setId
未能转换 com.ms.ui.IUIComponent.setIndeterminate
未能转换 com.ms.ui.IUIComponent.setInvalidating
未能转换 com.ms.ui.IUIComponent.setLocation(int, int)
未能转换 com.ms.ui.IUIComponent.setLocation(Point)
未能转换 com.ms.ui.IUIComponent.setPressed
未能转换 com.ms.ui.IUIComponent.setRedrawing
未能转换 com.ms.ui.IUIComponent.setSelected
未能转换 com.ms.ui.IUIComponent.setValid
未能转换 com.ms.ui.IUIComponent.setVisible
未能转换 com.ms.ui.IUIComponent.validate
未能转换 com.ms.ui.IUIContainer.add(IUIComponent)
未能转换 com.ms.ui.IUIContainer.add(IUIComponent, int)
未能转换 com.ms.ui.IUIContainer.add(IUIComponent, Object)
未能转换 com.ms.ui.IUIContainer.add(IUIComponent, Object, int)
未能转换 com.ms.ui.IUIContainer.add(String, IUIComponent)
未能转换 com.ms.ui.IUIContainer.adjustLayoutSize
未能转换 com.ms.ui.IUIContainer.continueInvalidate
未能转换 com.ms.ui.IUIContainer.ensureVisible
未能转换 com.ms.ui.IUIContainer.getChildBounds
未能转换 com.ms.ui.IUIContainer.getChildLocation
未能转换 com.ms.ui.IUIContainer.getChildSize
未能转换 com.ms.ui.IUIContainer.getComponentFromID
未能转换 com.ms.ui.IUIContainer.getEdge
未能转换 com.ms.ui.IUIContainer.getLayout
未能转换 com.ms.ui.IUIContainer.isOverlapping
未能转换 com.ms.ui.IUIContainer.navigate
未能转换 com.ms.ui.IUIContainer.paintComponents
未能转换 com.ms.ui.IUIContainer.passFocus
未能转换 com.ms.ui.IUIContainer.replace
未能转换 com.ms.ui.IUIContainer.setChildBounds

未能转换 com.ms.ui.IUIContainer.setChildLocation
未能转换 com.ms.ui.IUIContainer.setChildSize
未能转换 com.ms.ui.IUIContainer.setEdge
未能转换 com.ms.ui.IUIContainer.setHeader
未能转换 com.ms.ui.IUIContainer.setLayout
未能转换 com.ms.ui.IUILayoutManager
未能转换 com.ms.ui.IUIMenuLauncher
未能转换 com.ms.ui.IUIPosition
未能转换 com.ms.ui.IUIPropertyPage
未能转换 com.ms.ui.IUIRootContainer.componentMoved
未能转换 com.ms.ui.IUIRootContainer.endMenu
未能转换 com.ms.ui.IUIRootContainer.endTooltip
未能转换 com.ms.ui.IUIRootContainer.getFocus
未能转换 com.ms.ui.IUIRootContainer.getLaunchedMenu
未能转换 com.ms.ui.IUIRootContainer.launchMenu
未能转换 com.ms.ui.IUIRootContainer.launchTooltip
未能转换 com.ms.ui.IUIRootContainer.needsValidating
未能转换 com.ms.ui.IUIRootContainer.setFocus
未能转换 com.ms.ui.IUIScroll
未能转换 com.ms.ui.IUISelector
未能转换 com.ms.ui.IUISpinnerBuddy
未能转换 com.ms.ui.IUITree
未能转换 com.ms.ui.IUIWizardStep
未能转换 com.ms.ui.IWinEvent
未能转换 com.ms.ui.resource.DataBoundInputStream
未能转换 com.ms.ui.resource.ResourceDecoder
未能转换 com.ms.ui.resource.ResourceFormattingException
未能转换 com.ms.ui.resource.ResourceTypeListener
未能转换 com.ms.ui.resource.UIDialogLayout
未能转换 com.ms.ui.resource.Win32ResourceDecoder
未能转换 com.ms.ui.UIApplet.destroy
未能转换 com.ms.ui.UIApplet.getAppletContext
未能转换 com.ms.ui.UIApplet.getAudioClip(URL)
未能转换 com.ms.ui.UIApplet.getAudioClip(URL, String)
未能转换 com.ms.ui.UIApplet.getDocumentBase
未能转换 com.ms.ui.UIApplet.isActive
未能转换 com.ms.ui.UIApplet.play(URL)
未能转换 com.ms.ui.UIApplet.play(URL, String)
未能转换 com.ms.ui.UIApplet.setStub
未能转换 com.ms.ui.UIApplet.showStatus

未能转换 com.ms.ui.UIAwtHost.forwardEvents

未能转换 com.ms.ui.UIAwtHost.getCachedPreferredSize

未能转换 com.ms.ui.UIAwtHost.getMinimumSize

未能转换 com.ms.ui.UIAwtHost.getPreferredSize

未能转换 com.ms.ui.UIAwtHost.notifyEvent

未能转换 com.ms.ui.UIAwtHost.setFocused

未能转换 com.ms.ui.UIAwtHost.setValid

未能转换 com.ms.ui.UIBand

未能转换 com.ms.ui.UIBandBox

未能转换 com.ms.ui.UIBandThumb

未能转换 com.ms.ui.UIManager

未能转换 com.ms.ui.UIManager

未能转换 com.ms.ui.UIManager.doDefaultAction

未能转换 com.ms.ui.UIManager.getStyle

未能转换 com.ms.ui.UIManager.keyDown

未能转换 com.ms.ui.UIManager.keyUp

未能转换 com.ms.ui.UIManager.mouseClicked

未能转换 com.ms.ui.UIManager.setHot

未能转换 com.ms.ui.UIManager.setStyle

未能转换 com.ms.ui.UIManagerBar

未能转换 com.ms.ui.UIManagerBar.<ButtonBarType>

未能转换 com.ms.ui.UIManagerBar.add

未能转换 com.ms.ui.UIManagerBar.addTo

未能转换 com.ms.ui.UIManagerBar.UIManagerBar

未能转换 com.ms.ui.UICanvas.getID

未能转换 com.ms.ui.UICanvas.setID

未能转换 com.ms.ui.UICardLayout

未能转换 com.ms.ui.UICheckButton.getCheckImageSize

未能转换 com.ms.ui.UICheckButton.getConvertedEvent

未能转换 com.ms.ui.UICheckButton.getMinimumSize

未能转换 com.ms.ui.UICheckButton.getPreferredSize

未能转换 com.ms.ui.UICheckButton.paint

未能转换 com.ms.ui.UICheckButton.setCheckImageSize

未能转换 com.ms.ui.UICheckButton.setHot

未能转换 com.ms.ui.UICheckButton.setID

未能转换 com.ms.ui.UICheckButton.setPressed

未能转换 com.ms.ui.UICheckButton.setSelected

未能转换 com.ms.ui.UICheckGroup.add(IUIComponent)

未能转换 com.ms.ui.UICheckGroup.add(IUIComponent, int)

未能转换 com.ms.ui.UICheckGroup.add(IUIComponent, Object, int)

未能转换 com.ms.ui.UICheckGroup.add(String, Object, int)
未能转换 com.ms.ui.UICheckGroup.setHeader
未能转换 com.ms.ui.UICheckGroup.UICheckGroup
未能转换 com.ms.ui.UIChoice.add
未能转换 com.ms.ui.UIChoice.addSelectedIndex(int)
未能转换 com.ms.ui.UIChoice.addSelectedIndex(int, boolean)
未能转换 com.ms.ui.UIChoice.addSelectedIndices
未能转换 com.ms.ui.UIChoice.addSelectedItem
未能转换 com.ms.ui.UIChoice.addSelectedItems
未能转换 com.ms.ui.UIChoice.adjustPopupListSize
未能转换 com.ms.ui.UIChoice.getAnchorItem
未能转换 com.ms.ui.UIChoice.getConvertedEvent
未能转换 com.ms.ui.UIChoicegetExtensionItem
未能转换 com.ms.ui.UIChoice getMenu
未能转换 com.ms.ui.UIChoice.getPopupListMaxStringCount
未能转换 com.ms.ui.UIChoice.getPreferredSize
未能转换 com.ms.ui.UIChoice.getSelectedIndices
未能转换 com.ms.ui.UIChoice.getSelectedItem
未能转换 com.ms.ui.UIChoice.getSelectedItems
未能转换 com.ms.ui.UIChoice.getSelectionMode
未能转换 com.ms.ui.UIChoice.getStyle
未能转换 com.ms.ui.UIChoice.keyDown
未能转换 com.ms.ui.UIChoice.mouseDown
未能转换 com.ms.ui.UIChoice.remove
未能转换 com.ms.ui.UIChoice.removeSelectedIndex
未能转换 com.ms.ui.UIChoice.removeSelectedIndices
未能转换 com.ms.ui.UIChoice.removeSelectedItem
未能转换 com.ms.ui.UIChoice.removeSelectedItems
未能转换 com.ms.ui.UIChoice.setAnchorItem
未能转换 com.ms.ui.UIChoice.setExtensionItem
未能转换 com.ms.ui.UIChoice.setPopupListMaxStringCount
未能转换 com.ms.ui.UIChoice.setSelectedIndex(int)
未能转换 com.ms.ui.UIChoice.setSelectedIndex(int, boolean)
未能转换 com.ms.ui.UIChoice.setSelectedIndices(int[])
未能转换 com.ms.ui.UIChoice.setSelectedIndices(int[], boolean)
未能转换 com.ms.ui.UIChoice.setSelectedItem
未能转换 com.ms.ui.UIChoice.setSelectionMode
未能转换 com.ms.ui.UIChoice.setStyle
未能转换 com.ms.ui.UIChoice.setUsingWindowsLook
未能转换 com.ms.ui.UIChoice.THICK

未能转换 com.ms.ui.UIColorDialog.UIColorDialog
未能转换 com.ms.ui.UIColumnHeader.isMoving
未能转换 com.ms.ui.UIColumnHeader.isSizing
未能转换 com.ms.ui.UIColumnHeader.isSizingLeft
未能转换 com.ms.ui.UIColumnHeader.isSizingRight
未能转换 com.ms.ui.UIColumnHeader.mouseDown
未能转换 com.ms.ui.UIColumnViewer
未能转换 com.ms.ui.UIComponent.action
未能转换 com.ms.ui.UIComponent.adjustLayoutSize
未能转换 com.ms.ui.UIComponent.clone
未能转换 com.ms.ui.UIComponent.contains
未能转换 com.ms.ui.UIComponent.deliverEvent
未能转换 com.ms.ui.UIComponent.doDefaultAction
未能转换 com.ms.ui.UIComponent.doLayout
未能转换 com.ms.ui.UIComponent.ensureVisible
未能转换 com.ms.ui.UIComponent.getBounds
未能转换 com.ms.ui.UIComponent.getCachedPreferredSize
未能转换 com.ms.ui.UIComponent.getComponentAt
未能转换 com.ms.ui.UIComponent.getDefaultAction
未能转换 com.ms.ui.UIComponent.getDescription
未能转换 com.ms.ui.UIComponent.getGraphics
未能转换 com.ms.ui.UIComponent.getHelp
未能转换 com.ms.ui.UIComponent.getKeyboardShortcut
未能转换 com.ms.ui.UIComponent.getLocation
未能转换 com.ms.ui.UIComponent.getLocationOnScreen
未能转换 com.ms.ui.UIComponent.getMaximumSize
未能转换 com.ms.ui.UIComponent.getMinimumSize
未能转换 com.ms.ui.UIComponent.getPreferredSize
未能转换 com.ms.ui.UIComponent.getRoleCode
未能转换 com.ms.ui.UIComponent.getRoot
未能转换 com.ms.ui.UIComponent.getSize
未能转换 com.ms.ui.UIComponent.getStateCode
未能转换 com.ms.ui.UIComponent.getToolkit
未能转换 com.ms.ui.UIComponent.getTreeLock
未能转换 com.ms.ui.UIComponent.getValueText
未能转换 com.ms.ui.UIComponent.imageUpdate
未能转换 com.ms.ui.UIComponent.invalidate
未能转换 com.ms.ui.UIComponent.isChecked
未能转换 com.ms.ui.UIComponent.isEnabled
未能转换 com.ms.ui.UIComponent.isHeightRelative

未能转换 com.ms.ui.UIComponent.isHot
未能转换 com.ms.ui.UIComponent.isIndeterminate
未能转换 com.ms.ui.UIComponent.isInvalidating
未能转换 com.ms.ui.UIComponent.isKeyable
未能转换 com.ms.ui.UIComponent.isKeyable(boolean)
未能转换 com.ms.ui.UIComponent.isPressed
未能转换 com.ms.ui.UIComponent.isRedrawing
未能转换 com.ms.ui.UIComponent.isSelectable
未能转换 com.ms.ui.UIComponent.isShowing
未能转换 com.ms.ui.UIComponent.isValid
未能转换 com.ms.ui.UIComponent.isVisible
未能转换 com.ms.ui.UIComponent.isWidthRelative
未能转换 com.ms.ui.UIComponent.keyDown
未能转换 com.ms.ui.UIComponent.keyUp
未能转换 com.ms.ui.UIComponent.lostFocus
未能转换 com.ms.ui.UIComponent.mouseClicked
未能转换 com.ms.ui.UIComponent.mouseDown
未能转换 com.ms.ui.UIComponent.mouseDrag
未能转换 com.ms.ui.UIComponent.mouseEnter
未能转换 com.ms.ui.UIComponent.mouseExit
未能转换 com.ms.ui.UIComponent.mouseMove
未能转换 com.ms.ui.UIComponent.mouseUp
未能转换 com.ms.ui.UIComponent.navigate
未能转换 com.ms.ui.UIComponent.notifyEvent
未能转换 com.ms.ui.UIComponent.paint
未能转换 com.ms.ui.UIComponent.paintAll
未能转换 com.ms.ui.UIComponent.prepareImage(Image, ImageObserver)
未能转换 com.ms.ui.UIComponent.prepareImage(int, int, ImageObserver)
未能转换 com.ms.ui.UIComponent.print
未能转换 com.ms.ui.UIComponent.printAll
未能转换 com.ms.ui.UIComponent.recalcPreferredSize
未能转换 com.ms.ui.UIComponent.requestFocus
未能转换 com.ms.ui.UIComponent.repaint
未能转换 com.ms.ui.UIComponent.requestFocus
未能转换 com.ms.ui.UIComponent.setBounds
未能转换 com.ms.ui.UIComponent.setChecked
未能转换 com.ms.ui.UIComponent.setFlags
未能转换 com.ms.ui.UIComponent.setHot
未能转换 com.ms.ui.UIComponent.setId
未能转换 com.ms.ui.UIComponent.setIndeterminate

未能转换 com.ms.ui.UIComponent.setInvalidating
未能转换 com.ms.ui.UIComponent.setLocation(int, int)
未能转换 com.ms.ui.UIComponent.setLocation(Point)
未能转换 com.ms.ui.UIComponent.setPressed
未能转换 com.ms.ui.UIComponent.setRedrawing
未能转换 com.ms.ui.UIComponent.setSelected
未能转换 com.ms.ui.UIComponent.setSize
未能转换 com.ms.ui.UIComponent.setUsingWindowsLook
未能转换 com.ms.ui.UIComponent.setValid
未能转换 com.ms.ui.UIComponent.setValueText
未能转换 com.ms.ui.UIComponent.setVisible
未能转换 com.ms.ui.UIComponent.validate
未能转换 com.ms.ui.UIContainer.add(IUIComponent)
未能转换 com.ms.ui.UIContainer.add(IUIComponent, int)
未能转换 com.ms.ui.UIContainer.add(IUIComponent, Object)
未能转换 com.ms.ui.UIContainer.add(IUIComponent, Object, int)
未能转换 com.ms.ui.UIContainer.add(String, IUIComponent)
未能转换 com.ms.ui.UIContainer.adjustLayoutSize
未能转换 com.ms.ui.UIContainer.continueInvalidate
未能转换 com.ms.ui.UIContainer.ensureVisible
未能转换 com.ms.ui.UIContainer.getChildBounds
未能转换 com.ms.ui.UIContainer.getChildIndex
未能转换 com.ms.ui.UIContainer.getChildLocation
未能转换 com.ms.ui.UIContainer.getChildSize
未能转换 com.ms.ui.UIContainer.getClientRect
未能转换 com.ms.ui.UIContainer.getComponent
未能转换 com.ms.ui.UIContainer.getComponentFromID
未能转换 com.ms.ui.UIContainer.getComponentIndex
未能转换 com.ms.ui.UIContainer.getComponents
未能转换 com.ms.ui.UIContainer.getEdge
未能转换 com.ms.ui.UIContainer.getFocusComponent
未能转换 com.ms.ui.UIContainer.getID
未能转换 com.ms.ui.UIContainer.getInsets
未能转换 com.ms.ui.UIContainer.getLayout
未能转换 com.ms.ui.UIContainer.getMinimumSize
未能转换 com.ms.ui.UIContainer.getName
未能转换 com.ms.ui.UIContainer.getPreferredSize
未能转换 com.ms.ui.UIContainer.gotFocus
未能转换 com.ms.ui.UIContainer.invalidateAll
未能转换 com.ms.ui.UIContainer.isHeightRelative

未能转换 com.ms.ui.IContainer.isOverlapping
未能转换 com.ms.ui.IContainer.isWidthRelative
未能转换 com.ms.ui.IContainer.keyDown
未能转换 com.ms.ui.IContainer.lostFocus
未能转换 com.ms.ui.IContainer.mouseEnter
未能转换 com.ms.ui.IContainer.mouseExit
未能转换 com.ms.ui.IContainer.move(int, int)
未能转换 com.ms.ui.IContainer.move(IUIComponent, IUIComponent)
未能转换 com.ms.ui.IContainer.navigate
未能转换 com.ms.ui.IContainer.notifyEvent
未能转换 com.ms.ui.IContainer.paint
未能转换 com.ms.ui.IContainer.paintAll
未能转换 com.ms.ui.IContainer.paintComponents
未能转换 com.ms.ui.IContainer.passFocus
未能转换 com.ms.ui.IContainer.printAll
未能转换 com.ms.ui.IContainer.remove
未能转换 com.ms.ui.IContainer.removeAll
未能转换 com.ms.ui.IContainer.removeAllChildren
未能转换 com.ms.ui.IContainer.replace
未能转换 com.ms.ui.IContainer.setChildBounds
未能转换 com.ms.ui.IContainer.setChildLocation
未能转换 com.ms.ui.IContainer.setChildSize
未能转换 com.ms.ui.IContainer.setComponent
未能转换 com.ms.ui.IContainer.setEdge
未能转换 com.ms.ui.IContainer.setId
未能转换 com.ms.ui.IContainer.setLayout
未能转换 com.ms.ui.IContainer.setLocation
未能转换 com.ms.ui.IContainer.setSize
未能转换 com.ms.ui.UIContextMenu.action
未能转换 com.ms.ui.UIContextMenu.ended
未能转换 com.ms.ui.UIContextMenu.getPlacement
未能转换 com.ms.ui.UIContextMenu.getRoot
未能转换 com.ms.ui.UIContextMenu.getUserItem
未能转换 com.ms.ui.UIContextMenu.UIContextMenu
未能转换 com.ms.ui.UIDialog.adjustLayoutSize
未能转换 com.ms.ui.UIDialog.isAutoPack
未能转换 com.ms.ui.UIDialog.keyDown
未能转换 com.ms.ui.UIDialog.keyUp
未能转换 com.ms.ui.UIDialog.position
未能转换 com.ms.ui.UIDialog.setAutoPack

未能转换 com.ms.ui.UIDialog.setModal
未能转换 com.ms.ui.UIDialog.UIDialog(UIFrame, boolean)
未能转换 com.ms.ui.UIDialog.UIDialog(UIFrame, String, boolean)
未能转换 com.ms.ui.UIDialogMapping
未能转换 com.ms.ui.UIDragDrop
未能转换 com.ms.ui.UIDrawText.ensureNotDirty
未能转换 com.ms.ui.UIDrawText.ensureVisible(Point)
未能转换 com.ms.ui.UIDrawText.ensureVisible(Rectangle)
未能转换 com.ms.ui.UIDrawText.eoln
未能转换 com.ms.ui.UIDrawText.getCachedGraphics
未能转换 com.ms.ui.UIDrawText.getCaretWidth
未能转换 com.ms.ui.UIDrawText.getCharFromScreen(int, int)
未能转换 com.ms.ui.UIDrawText.getCharFromScreen(Point)
未能转换 com.ms.ui.UIDrawText.getCharLocation
未能转换 com.ms.ui.UIDrawText.getMinimumSize
未能转换 com.ms.ui.UIDrawText.getOutline
未能转换 com.ms.ui.UIDrawText.getPreferredSize
未能转换 com.ms.ui.UIDrawText.getStartPoint
未能转换 com.ms.ui.UIDrawText.getVertAlign
未能转换 com.ms.ui.UIDrawText.getWordEdge
未能转换 com.ms.ui.UIDrawText.requestFocus
未能转换 com.ms.ui.UIDrawText.hideCaret
未能转换 com.ms.ui.UIDrawText.isAutoResizable
未能转换 com.ms.ui.UIDrawText_KeyDown
未能转换 com.ms.ui.UIDrawText_LostFocus
未能转换 com.ms.ui.UIDrawText_MouseDown
未能转换 com.ms.ui.UIDrawText_MouseDrag
未能转换 com.ms.ui.UIDrawText_paint(FxGraphics)
未能转换 com.ms.ui.UIDrawText_paint(FxGraphics, int, int, boolean)
未能转换 com.ms.ui.UIDrawText_setAutoResizable
未能转换 com.ms.ui.UIDrawText_setCaretWidth
未能转换 com.ms.ui.UIDrawText_setCurrIndex
未能转换 com.ms.ui.UIDrawText_setHorizAlign
未能转换 com.ms.ui.UIDrawText_setInputMethod
未能转换 com.ms.ui.UIDrawText_setOutline
未能转换 com.ms.ui.UIDrawText_setRefresh
未能转换 com.ms.ui.UIDrawText_setTabs
未能转换 com.ms.ui.UIDrawText_setTextCallback
未能转换 com.ms.ui.UIDrawText_setUnderlined
未能转换 com.ms.ui.UIDrawText_setVertAlign

未能转换 com.ms.ui.UIDrawText.setWordWrap
未能转换 com.ms.ui.UIDrawText.showCaret
未能转换 com.ms.ui.UIEdit.allowUndo
未能转换 com.ms.ui.UIEdit.append(char, boolean)
未能转换 com.ms.ui.UIEdit.append(char[], boolean)
未能转换 com.ms.ui.UIEdit.append(String, boolean)
未能转换 com.ms.ui.UIEdit.clear
未能转换 com.ms.ui.UIEdit.EXCLUDE
未能转换 com.ms.ui.UIEdit.getConvertedEvent
未能转换 com.ms.ui.UIEdit.getMaskChars
未能转换 com.ms.ui.UIEdit.getMaskMode
未能转换 com.ms.ui.UIEdit.INCLUDE
未能转换 com.ms.ui.UIEdit.insert(char, int, boolean)
未能转换 com.ms.ui.UIEdit.insert(char[], int, boolean)
未能转换 com.ms.ui.UIEdit.insert(String, int, boolean)
未能转换 com.ms.ui.UIEdit.isRedoable
未能转换 com.ms.ui.UIEdit.isUndoable
未能转换 com.ms.ui.UIEdit.isUndoAllowed
未能转换 com.ms.ui.UIEdit.keyDown
未能转换 com.ms.ui.UIEdit.READONLY
未能转换 com.ms.ui.UIEdit.redo
未能转换 com.ms.ui.UIEdit.redo(boolean)
未能转换 com.ms.ui.UIEdit.remove
未能转换 com.ms.ui.UIEdit.setMaskChars
未能转换 com.ms.ui.UIEdit.setMaskMode
未能转换 com.ms.ui.UIEditChoice.action
未能转换 com.ms.ui.UIEditChoice.getEditComponent
未能转换 com.ms.ui.UIEditChoice.keyDown
未能转换 com.ms.ui.UIEditChoice.lostFocus
未能转换 com.ms.ui.UIExpandButton
未能转换 com.ms.ui.UIFindReplaceDialog
未能转换 com.ms.ui.UIFixedFlowLayout
未能转换 com.ms.ui.UIFixedFlowLayout.computeUnitDimension
未能转换 com.ms.ui.UIFixedFlowLayout.getAlignment
未能转换 com.ms.ui.UIFixedFlowLayout.getPreferredSize
未能转换 com.ms.ui.UIFixedFlowLayout.isOverlapping
未能转换 com.ms.ui.UIFixedFlowLayout.isWidthRelative
未能转换 com.ms.ui.UIFixedFlowLayout.layout
未能转换 com.ms.ui.UIFixedFlowLayout.navigate

未能转换 com.ms.ui.UIFixedFlowLayout.setAlignment
未能转换 com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout
未能转换 com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout(int, int)
未能转换 com.ms.ui.UIFixedFlowLayout.UIFixedFlowLayout(int, int, int)
未能转换 com.ms.ui.UIFlowLayout
未能转换 com.ms.ui.UIFlowLayout.getAlignment
未能转换 com.ms.ui.UIFlowLayout.getMinimumSize
未能转换 com.ms.ui.UIFlowLayout.getPreferredSize
未能转换 com.ms.ui.UIFlowLayout.isOverlapping
未能转换 com.ms.ui.UIFlowLayout.isWidthRelative
未能转换 com.ms.ui.UIFlowLayout.layout
未能转换 com.ms.ui.UIFlowLayout.navigate
未能转换 com.ms.ui.UIFlowLayout.setAlignment
未能转换 com.ms.ui.UIFlowLayout.setWrap
未能转换 com.ms.ui.UIFlowLayout.UIFlowLayout
未能转换 com.ms.ui.UIFlowLayout.UIFlowLayout(int)
未能转换 com.ms.ui.UIFlowLayout.UIFlowLayout(int, int, int)
未能转换 com.ms.ui.UIFontDialog.UIFontDialog(UIFrame)
未能转换 com.ms.ui.UIFontDialog.UIFontDialog(UIFrame, String[])
未能转换 com.ms.ui.UIFrame.setMenuBar
未能转换 com.ms.ui.UIGraphic.getPreferredSize
未能转换 com.ms.ui.UIGraphic.imageUpdate
未能转换 com.ms.ui.UIGraphic.paint
未能转换 com.ms.ui.UIGraphic.UIGraphic
未能转换 com.ms.ui.UIManagerBagConstraints
未能转换 com.ms.ui.UIManagerBagLayout
未能转换 com.ms.ui.UIManagerLayout
未能转换 com.ms.ui.UIManagerLayout.continueInvalidate
未能转换 com.ms.ui.UIManagerLayout.getMinimumSize
未能转换 com.ms.ui.UIManagerLayout.getPreferredSize
未能转换 com.ms.ui.UIManagerLayout.isOverlapping
未能转换 com.ms.ui.UIManagerLayout.layout
未能转换 com.ms.ui.UIManagerLayout.navigate
未能转换 com.ms.ui.UIManagerLayout.UIManagerLayout(int, int)
未能转换 com.ms.ui.UIManagerLayout.UIManagerLayout(int, int, int, int)
未能转换 com.ms.ui.UIGroup.adjustLayoutSize
未能转换 com.ms.ui.UIGroup.paint
未能转换 com.ms.ui.UIGroup.UIGroup
未能转换 com.ms.ui.UIHeaderRow.mouseDown
未能转换 com.ms.ui.UIHeaderRow.mouseDrag

未能转换 com.ms.ui.UIHeaderRow.mouseUp
未能转换 com.ms.ui.UIHeaderRow.navigate
未能转换 com.ms.ui.UILItem.<Position>
未能转换 com.ms.ui.UILItem.getGap
未能转换 com.ms.ui.UILItem.getImagePos
未能转换 com.ms.ui.UILItem.getPreferredSize
未能转换 com.ms.ui.UILItem.imageUpdate
未能转换 com.ms.ui.UILItem.paint
未能转换 com.ms.ui.UILItem.setFocused
未能转换 com.ms.ui.UILItem.setGap
未能转换 com.ms.ui.UILItem.setHot
未能转换 com.ms.ui.UILItem.setImagePos
未能转换 com.ms.ui.UILItem.setSelected
未能转换 com.ms.ui.UILItem.UILItem(Image, String, int)
未能转换 com.ms.ui.UILItem.UILItem(Image, String, int, int)
未能转换 com.ms.uiUILayoutManager
未能转换 com.ms.uiUILine.getMinimumSize
未能转换 com.ms.uiUILine.getPreferredSize
未能转换 com.ms.uiUILine.paint
未能转换 com.ms.uiUIList.UILList(int)
未能转换 com.ms.uiUIList.UILList(int, int)
未能转换 com.ms.uiUIMarquee
未能转换 com.ms.uiUIMenuButton.action
未能转换 com.ms.uiUIMenuButton.ended
未能转换 com.ms.uiUIMenuButton.getPlacement
未能转换 com.ms.uiUIMenuButton.keyDown
未能转换 com.ms.uiUIMenuButton.launch
未能转换 com.ms.uiUIMenuButton.mouseDown
未能转换 com.ms.uiUIMenuButton.requestFocus
未能转换 com.ms.uiUIMenuButton.UIMenuButton(IUIComponent, int, UIMenuList)
未能转换 com.ms.uiUIMenuButton.UIMenuButton(IUIComponent, UIMenuList)
未能转换 com.ms.uiUIMenuButton.UIMenuButton(String, int, UIMenuList)
未能转换 com.ms.uiUIMenulItem.UIMenulItem
未能转换 com.ms.uiUIMenulItem.action
未能转换 com.ms.uiUIMenulItem.getInsets
未能转换 com.ms.uiUIMenulItem.keyDown
未能转换 com.ms.uiUIMenulItem.paint
未能转换 com.ms.uiUIMenulItem.UIMenulItem(IUIComponent)
未能转换 com.ms.uiUIMenulItem.UIMenulItem(IUIComponent, UIMenuList)
未能转换 com.ms.uiUIMenuLauncher.cancel

未能转换 com.ms.ui.UIMenuLauncher.ended
未能转换 com.ms.ui.UIMenuLauncher.fitToScreen
未能转换 com.ms.ui.UIMenuLauncher.getDisplayer
未能转换 com.ms.ui.UIMenuLauncher.getPlacement
未能转换 com.ms.ui.UIMenuLauncher.isLaunched
未能转换 com.ms.ui.UIMenuLauncher.launch
未能转换 com.ms.ui.UIMenuLauncher.raiseEvent
未能转换 com.ms.ui.UIMenuLauncher.setFocused
未能转换 com.ms.ui.UIMenuLauncher.setHot
未能转换 com.ms.ui.UIMenuLauncher.setSelected
未能转换 com.ms.ui.UIMenuLauncher.UIMenuLauncher(IUIComponent)
未能转换 com.ms.ui.UIMenuLauncher.UIMenuLauncher(IUIComponent, UIMenuList)
未能转换 com.ms.ui.UIMenuLauncher.UIMenuLauncher(UIMenuList)
未能转换 com.ms.ui.UIMenuList
未能转换 com.ms.ui.UIMenuList.action
未能转换 com.ms.ui.UIMenuList getMenuLauncher
未能转换 com.ms.ui.UIMenuList.gotFocus
未能转换 com.ms.ui.UIMenuList.keyDown
未能转换 com.ms.ui.UIMenuList.lostFocus
未能转换 com.ms.ui.UIMenuList.mouseClicked
未能转换 com.ms.ui.UIMenuList.mouseEnter
未能转换 com.ms.ui.UIMenuList.requestFocus
未能转换 com.ms.ui.UIMenuList.setMenuLauncher
未能转换 com.ms.ui.UIMenuList.UIMenuList
未能转换 com.ms.ui.UIMessageBox
未能转换 com.ms.ui.UIMessageBox.action
未能转换 com.ms.ui.UIMessageBox.doModal
未能转换 com.ms.ui.UIMessageBox.doModalIndex
未能转换 com.ms.ui.UIMessageBox.getButtonAlignment
未能转换 com.ms.ui.UIMessageBox.getButtons
未能转换 com.ms.ui.UIMessageBox.getDefaultButton
未能转换 com.ms.ui.UIMessageBox.getFrame
未能转换 com.ms.ui.UIMessageBox.getImage
未能转换 com.ms.ui.UIMessageBox.getInsets
未能转换 com.ms.ui.UIMessageBox.getPreferredSize
未能转换 com.ms.ui.UIMessageBox.getText
未能转换 com.ms.ui.UIMessageBox.getTimeout
未能转换 com.ms.ui.UIMessageBox Insets
未能转换 com.ms.ui.UIMessageBox.keyDown
未能转换 com.ms.ui.UIMessageBox.keyUp

未能转换 com.ms.ui.UIMessageBox.setButtonAlignment
未能转换 com.ms.ui.UIMessageBox.setButtons
未能转换 com.ms.ui.UIMessageBox.setDefaultButton
未能转换 com.ms.ui.UIMessageBox.setImage
未能转换 com.ms.ui.UIMessageBox.setText
未能转换 com.ms.ui.UIMessageBox.setTimeout
未能转换 com.ms.ui.UIMessageBox.timeTriggered
未能转换 com.ms.ui.UIMessageBox.UIMessageBox(UIFrame)
未能转换 com.ms.ui.UIMessageBox.UIMessageBox(UIFrame, String, String, int, int)
未能转换 com.ms.ui.UIMSVMPopup
未能转换 com.ms.ui.UIOldEvent
未能转换 com.ms.ui.UIPanel.add
未能转换 com.ms.ui.UIPanel.getChild
未能转换 com.ms.ui.UIPanel.getChildCount
未能转换 com.ms.ui.UIPanel.getHeader
未能转换 com.ms.ui.UIPanel.getId
未能转换 com.ms.ui.UIPanel.getLayout
未能转换 com.ms.ui.UIPanel.paintAll
未能转换 com.ms.ui.UIPanel.setId
未能转换 com.ms.ui.UIPanel.setLayout
未能转换 com.ms.ui.UIPanel.setRedrawing
未能转换 com.ms.ui.UIPanel.UIPanel
未能转换 com.ms.ui.UIProgress.paint
未能转换 com.ms.ui.UIProgress.UIProgress
未能转换 com.ms.ui.UIProgress.UIProgress(int)
未能转换 com.ms.ui.UIProgress.UIProgress(int, int)
未能转换 com.ms.ui.UIProgress.update
未能转换 com.ms.ui.UIPropertyDialog
未能转换 com.ms.ui.UIPropertyPage
未能转换 com.ms.ui.UIPushButton.getConvertedEvent
未能转换 com.ms.ui.UIPushButton.paint
未能转换 com.ms.ui.UIPushButton.setChecked
未能转换 com.ms.ui.UIPushButton.setFocused
未能转换 com.ms.ui.UIPushButton.setPressed
未能转换 com.ms.ui.UIRadioButton.UIRadioButton(IUIComponent, int)
未能转换 com.ms.ui.UIRadioButton.UIRadioButton(String, int)
未能转换 com.ms.ui.UIRadioGroup.add
未能转换 com.ms.ui.UIRadioGroup.UIRadioGroup
未能转换 com.ms.ui.UIRepeatButton.mouseClicked
未能转换 com.ms.ui.UIRepeatButton.setPressed

未能转换 com.ms.ui.UIRepeatButton.timeTriggered
未能转换 com.ms.ui.UIRepeatButton.UIRepeatButton
未能转换 com.ms.ui.UIRepeatButton.UIRepeatButton(IUIComponent)
未能转换 com.ms.ui.UIRepeatButton.UIRepeatButton(IUIComponent, int)
未能转换 com.ms.ui.UIRepeatButton.UIRepeatButton(String)
未能转换 com.ms.ui.UIRepeatButton.UIRepeatButton(String, int)
未能转换 com.ms.ui.UIRow.getName
未能转换 com.ms.ui.UIRow.requestFocus
未能转换 com.ms.ui.UIRow.setSelected
未能转换 com.ms.ui.UIRowLayout
未能转换 com.ms.ui.UIScroll
未能转换 com.ms.ui.UIScroll.scrollLineDown
未能转换 com.ms.ui.UIScroll.scrollLineUp
未能转换 com.ms.ui.UIScroll.scrollPageDown
未能转换 com.ms.ui.UIScroll.scrollPageUp
未能转换 com.ms.ui.UIScroll.setScrollInfo
未能转换 com.ms.ui.UIScroll.setScrollLine
未能转换 com.ms.ui.UIScrollBar
未能转换 com.ms.ui.UIScrollBar.<Direction>
未能转换 com.ms.ui.UIScrollBar.add
未能转换 com.ms.ui.UIScrollBar.getLayoutComponent
未能转换 com.ms.ui.UIScrollBar.setStyle
未能转换 com.ms.ui.UIScrollBar.setUsingWindowsLook
未能转换 com.ms.ui.UIScrollBar.UIScrollBar
未能转换 com.ms.ui.UIScrollBar.UIScrollBar(int)
未能转换 com.ms.ui.UIScrollBar.UIScrollBar(int, int, int, int, int, int)
未能转换 com.ms.ui.UIScrollThumb
未能转换 com.ms.ui.UIScrollView
未能转换 com.ms.ui.UIScrollView.<Position>
未能转换 com.ms.ui.UIScrollView.add
未能转换 com.ms.ui.UIScrollView.CONT
未能转换 com.ms.ui.UIScrollView.CONTENT
未能转换 com.ms.ui.UIScrollView.ensureVisible
未能转换 com.ms.ui.UIScrollView.getContent
未能转换 com.ms.ui.UIScrollView.getHLine
未能转换 com.ms.ui.UIScrollView.getLayoutComponent
未能转换 com.ms.ui.UIScrollView.getLine
未能转换 com.ms.ui.UIScrollView.getMinimumSize
未能转换 com.ms.ui.UIScrollView.getPosition
未能转换 com.ms.ui.UIScrollView.getPreferredSize

未能转换 com.ms.ui.UIScrollViewer.getRoleCode
未能转换 com.ms.ui.UIScrollViewer.getVLine
未能转换 com.ms.ui.UIScrollViewer.getXPosition
未能转换 com.ms.ui.UIScrollViewer.getYPosition
未能转换 com.ms.ui.UIScrollViewer.isKeyable
未能转换 com.ms.ui.UIScrollViewer.keyDown
未能转换 com.ms.ui.UIScrollViewer.remove
未能转换 com.ms.ui.UIScrollViewer.replace
未能转换 com.ms.ui.UIScrollViewer.setContent
未能转换 com.ms.ui.UIScrollViewer.setHLine
未能转换 com.ms.ui.UIScrollViewer.setLine
未能转换 com.ms.ui.UIScrollViewer.setScrollStyle
未能转换 com.ms.ui.UIScrollViewer.setUsingWindowsLook
未能转换 com.ms.ui.UIScrollViewer.setVLine
未能转换 com.ms.ui.UIScrollViewer.setXPosition
未能转换 com.ms.ui.UIScrollViewer.setYPosition
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent)
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent, int)
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent, int, int)
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent, int, int, int)
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent, int, int, int, int)
未能转换 com.ms.ui.UIScrollViewer.UIScrollViewer(IUIComponent, int, int, int, int, int)
未能转换 com.ms.ui.UISelector
未能转换 com.ms.ui.UISelector.addSelectedIndex(int)
未能转换 com.ms.ui.UISelector.addSelectedIndex(int, boolean)
未能转换 com.ms.ui.UISelector.addSelectedIndices(int[])
未能转换 com.ms.ui.UISelector.addSelectedIndices(int[], boolean)
未能转换 com.ms.ui.UISelector.find
未能转换 com.ms.ui.UISelector.getSelectedIndex
未能转换 com.ms.ui.UISelector.getSelectedItem
未能转换 com.ms.ui.UISelector.remove
未能转换 com.ms.ui.UISelector.removeSelectedIndex(int)
未能转换 com.ms.ui.UISelector.removeSelectedIndex(int, boolean)
未能转换 com.ms.ui.UISelector.removeSelectedIndices(int[])
未能转换 com.ms.ui.UISelector.removeSelectedIndices(int[], boolean)
未能转换 com.ms.ui.UISelector.setSelectedIndex(int)
未能转换 com.ms.ui.UISelector.setSelectedIndex(int, boolean)
未能转换 com.ms.ui.UISelector.setSelectedIndices(int[])
未能转换 com.ms.ui.UISelector.setSelectedIndices(int[], boolean)

未能转换 com.ms.ui.UISelector.setSelectedItem
未能转换 com.ms.ui.UISelector.setSelectionMode
未能转换 com.ms.ui.UISingleContainer.add
未能转换 com.ms.ui.UISingleContainer.getChildBounds
未能转换 com.ms.ui.UISingleContainer.getChildLocation
未能转换 com.ms.ui.UISingleContainer.getChildSize
未能转换 com.ms.ui.UISingleContainer.getComponent
未能转换 com.ms.ui.UISingleContainer.getHeader
未能转换 com.ms.ui.UISingleContainer.layout
未能转换 com.ms.ui.UISingleContainer.mouseExit
未能转换 com.ms.ui.UISingleContainer.relayout
未能转换 com.ms.ui.UISingleContainer.remove
未能转换 com.ms.ui.UISingleContainer.setHeader
未能转换 com.ms.ui.UISingleContainer.setId
未能转换 com.ms.ui.UISingleContainer.setName
未能转换 com.ms.ui.UISingleContainer.UISingleContainer
未能转换 com.ms.ui.UISlider.add
未能转换 com.ms.ui.UISlider.clearSelection
未能转换 com.ms.ui.UISlider.getLayoutComponent
未能转换 com.ms.ui.UISlider.getSelectionEnd
未能转换 com.ms.ui.UISlider.getSelectionStart
未能转换 com.ms.ui.UISlider.setSelection
未能转换 com.ms.ui.UISlider.setSelectionEnd
未能转换 com.ms.ui.UISlider.setSelectionStart
未能转换 com.ms.ui.UISpinner
未能转换 com.ms.ui.UISpinner.<Direction>
未能转换 com.ms.ui.UISpinner.<Type>
未能转换 com.ms.ui.UISpinner.add
未能转换 com.ms.ui.UISpinner.getLayoutComponent
未能转换 com.ms.ui.UISpinner.getMinimumSize
未能转换 com.ms.ui.UISpinner.getPreferredSize
未能转换 com.ms.ui.UISpinner.getRoleCode
未能转换 com.ms.ui.UISpinner.getScrollPos
未能转换 com.ms.ui.UISpinner.keyDown
未能转换 com.ms.ui.UISpinner.layout
未能转换 com.ms.ui.UISpinner.RAISED
未能转换 com.ms.ui.UISpinner.requestFocus
未能转换 com.ms.ui.UISpinner.scrollLineDown
未能转换 com.ms.ui.UISpinner.scrollLineUp
未能转换 com.ms.ui.UISpinner.scrollPageDown

未能转换 com.ms.ui.UISpinner.scrollPageUp
未能转换 com.ms.ui.UISpinner.setScrollInfo
未能转换 com.ms.ui.UISpinner.setStyle
未能转换 com.ms.ui.UISpinner.UISpinner(int)
未能转换 com.ms.ui.UISpinner.UISpinner(int, int, int, int, int, int)
未能转换 com.ms.ui.UISpinnerEdit.BORDER
未能转换 com.ms.ui.UISpinnerEdit.CENTER
未能转换 com.ms.ui.UISpinnerEdit.getEditStyle
未能转换 com.ms.ui.UISpinnerEdit.setEditable
未能转换 com.ms.ui.UISpinnerEdit.UISpinnerEdit(int, int)
未能转换 com.ms.ui.UISpinnerEdit.UISpinnerEdit(int, int, int, int, int, int)
未能转换 com.ms.ui.UISplitLayout
未能转换 com.ms.ui.UISplitViewer
未能转换 com.ms.ui.UISplitViewer.<Type>
未能转换 com.ms.ui.UISplitViewer.getComponent
未能转换 com.ms.ui.UISplitViewer.getFloater
未能转换 com.ms.ui.UISplitViewer.getPos
未能转换 com.ms.ui.UISplitViewer.getStyle
未能转换 com.ms.ui.UISplitViewer.mouseDown
未能转换 com.ms.ui.UISplitViewer.mouseDrag
未能转换 com.ms.ui.UISplitViewer.mouseUp
未能转换 com.ms.ui.UISplitViewer.setPos
未能转换 com.ms.ui.UISplitViewer.UISplitViewer
未能转换 com.ms.ui.UIStateComponent.adjustLayoutSize
未能转换 com.ms.ui.UIStateComponent.disableEvents
未能转换 com.ms.ui.UIStateComponent.enableEvents
未能转换 com.ms.ui.UIStateComponent.getCachedPreferredSize
未能转换 com.ms.ui.UIStateComponent.getConvertedEvent
未能转换 com.ms.ui.UIStateComponent.getFlags
未能转换 com.ms.ui.UIStateComponent.HEADER
未能转换 com.ms.ui.UIStateComponent.isInValidating
未能转换 com.ms.ui.UIStateComponent.isRedrawing
未能转换 com.ms.ui.UIStateComponent.isSelected
未能转换 com.ms.ui.UIStateComponent.isValid
未能转换 com.ms.ui.UIStateComponent.listeners
未能转换 com.ms.ui.UIStateComponent.obtainListenerTracker
未能转换 com.ms.ui.UIStateComponent.recalcPreferredSize
未能转换 com.ms.ui.UIStateComponent.setChecked
未能转换 com.ms.ui.UIStateComponent.setFlags
未能转换 com.ms.ui.UIStateComponent.setHot

未能转换 com.ms.ui.UIStateComponent.setIndeterminate
未能转换 com.ms.ui.UIStateComponent.setInvalidating
未能转换 com.ms.ui.UIStateComponent.setPressed
未能转换 com.ms.ui.UIStateComponent.setRedrawing
未能转换 com.ms.ui.UIStateComponent.setReparent
未能转换 com.ms.ui.UIStateComponent.setSelected
未能转换 com.ms.ui.UIStateComponent.setUsingWindowsLook
未能转换 com.ms.ui.UIStateComponent.setValid
未能转换 com.ms.ui.UIStateContainer.add
未能转换 com.ms.ui.UIStateContainer.adjustLayoutSize
未能转换 com.ms.ui.UIStateContainer.disableEvents
未能转换 com.ms.ui.UIStateContainer.enableEvents
未能转换 com.ms.ui.UIStateContainer.getBackground
未能转换 com.ms.ui.UIStateContainer.getPreferredSize
未能转换 com.ms.ui.UIStateContainer.getConvertedEvent
未能转换 com.ms.ui.UIStateContainer.getCursor
未能转换 com.ms.ui.UIStateContainer.getEdge
未能转换 com.ms.ui.UIStateContainer.getFlags
未能转换 com.ms.ui.UIStateContainer.getFont
未能转换 com.ms.ui.UIStateContainer.getForeground
未能转换 com.ms.ui.UIStateContainer.getIndex
未能转换 com.ms.ui.UIStateContainer.getParent
未能转换 com.ms.ui.UIStateContainer.isChecked
未能转换 com.ms.ui.UIStateContainer.isEnabled
未能转换 com.ms.ui.UIStateContainer.isFocused
未能转换 com.ms.ui.UIStateContainer.isInvalidating
未能转换 com.ms.ui.UIStateContainer.isRedrawing
未能转换 com.ms.ui.UIStateContainer.isSelected
未能转换 com.ms.ui.UIStateContainer.isValid
未能转换 com.ms.ui.UIStateContainer.isVisible
未能转换 com.ms.ui.UIStateContainer.listeners
未能转换 com.ms.ui.UIStateContainer.obtainConvertedEvent
未能转换 com.ms.ui.UIStateContainer.obtainListenerTracker
未能转换 com.ms.ui.UIStateContainer.recalcPreferredSize
未能转换 com.ms.ui.UIStateContainer.setBackground
未能转换 com.ms.ui.UIStateContainer.setChecked
未能转换 com.ms.ui.UIStateContainer.setCursor
未能转换 com.ms.ui.UIStateContainer.setEdge
未能转换 com.ms.ui.UIStateContainer.setEnabled
未能转换 com.ms.ui.UIStateContainer.setFlags

未能转换 com.ms.ui.UIStateContainer.setFont
未能转换 com.ms.ui.UIStateContainer.setForeground
未能转换 com.ms.ui.UIStateContainer.setHot
未能转换 com.ms.ui.UIStateContainer.setIndeterminate
未能转换 com.ms.ui.UIStateContainer.setIndex
未能转换 com.ms.ui.UIStateContainer.setInvalidating
未能转换 com.ms.ui.UIStateContainer.setParent
未能转换 com.ms.ui.UIStateContainer.setPressed
未能转换 com.ms.ui.UIStateContainer.setRedrawing
未能转换 com.ms.ui.UIStateContainer.setReparent
未能转换 com.ms.ui.UIStateContainer.setSelected
未能转换 com.ms.ui.UIStateContainer.setUsingWindowsLook
未能转换 com.ms.ui.UIStateContainer.setValid
未能转换 com.ms.ui.UIStateContainer.setVisible
未能转换 com.ms.ui.UIStateContainer.UIStateContainer
未能转换 com.ms.ui.UIStatic
未能转换 com.ms.ui.UIStatic.HCENTER
未能转换 com.ms.ui.UIStatic.setFlags
未能转换 com.ms.ui.UIStatic.VCENTER
未能转换 com.ms.ui.UISystem
未能转换 com.ms.ui.UITab.paint
未能转换 com.ms.ui.UITab.setSelected
未能转换 com.ms.ui.UITab.UITab
未能转换 com.ms.ui.UITabLayout
未能转换 com.ms.ui.UITabList
未能转换 com.ms.ui.UITabList.add(IUIComponent)
未能转换 com.ms.ui.UITabList.add(IUIComponent, int)
未能转换 com.ms.ui.UITabList.add(String)
未能转换 com.ms.ui.UITabList.add(String, int)
未能转换 com.ms.ui.UITabList.layout
未能转换 com.ms.ui.UITabList.paint
未能转换 com.ms.ui.UITabList.passFocus
未能转换 com.ms.ui.UITabList.setSelectedItem
未能转换 com.ms.ui.UITabListLayout
未能转换 com.ms.ui.UITabViewer.add
未能转换 com.ms.ui.UITabViewer.addTab(IUIComponent, IUIComponent)
未能转换 com.ms.ui.UITabViewer.getConvertedEvent
未能转换 com.ms.ui.UITabViewer.paint
未能转换 com.ms.ui.UITabViewer.removeTab
未能转换 com.ms.ui.UITabViewer.requestFocus

未能转换 com.ms.ui.UITabViewer.setSelectedItem
未能转换 com.ms.ui.UIText.getPreferredSize
未能转换 com.ms.ui.UIText.paint
未能转换 com.ms.ui.UIText.setFocused
未能转换 com.ms.ui.UIText.setHot
未能转换 com.ms.ui.UIText.setSelected
未能转换 com.ms.ui.UIThreePanelLayout
未能转换 com.ms.ui.UIThumb
未能转换 com.ms.ui.UITree
未能转换 com.ms.ui.UITree.action
未能转换 com.ms.ui.UITree.add(Component)
未能转换 com.ms.ui.UITree.add(Component, int)
未能转换 com.ms.ui.UITree.add(Image, String, int)
未能转换 com.ms.ui.UITree.add(String, int)
未能转换 com.ms.ui.UITree.add(IUIComponent)
未能转换 com.ms.ui.UITree.getAttachRect
未能转换 com.ms.ui.UITree.getExpander
未能转换 com.ms.ui.UITree.keyDown
未能转换 com.ms.ui.UITree.remove
未能转换 com.ms.ui.UITree.setChecked
未能转换 com.ms.ui.UITree.setExpanded
未能转换 com.ms.ui.UITree.setExpander
未能转换 com.ms.ui.UITree.setLayout
未能转换 com.ms.ui.UITreeLayout
未能转换 com.ms.ui.UIVerticalFlowLayout
未能转换 com.ms.ui.UIViewer.ensureVisible
未能转换 com.ms.ui.UIViewer.isOverlapping
未能转换 com.ms.ui.UIViewer.mouseDown
未能转换 com.ms.ui.UIViewer.mouseDrag
未能转换 com.ms.ui.UIViewer.mouseUp
未能转换 com.ms.ui.UIViewer.requestFocus
未能转换 com.ms.ui.UIViewer.timeTriggered
未能转换 com.ms.ui.UIWindow_KeyDown
未能转换 com.ms.ui.UIWindow_Pack
未能转换 com.ms.ui.UIWindow_SetSize
未能转换 com.ms.ui.UIWinEvent
未能转换 com.ms.ui.UIWizard
未能转换 com.ms.ui.UIWizardStep
未能转换 com.ms.ui.Version

Com.ms.util 错误信息

未能转换 com.ms.util.ArraySort.compare

未能转换 com.ms.util.ArraySort.sort

未能转换 com.ms.util.ArraySort.swap

未能转换 com.ms.util.cab.CabEnumerator

未能转换 com.ms.util.cab.CabException

未能转换 com.ms.util.cab.CabFileEntry

未能转换 com.ms.util.cab.CabFolderEntry

未能转换 com.ms.util.cab.CabProgressInterface

未能转换 com.ms.util.EventLog.reportEvent

未能转换 com.ms.util.HTMLTokenizer

未能转换 com.ms.util.IIntRangeComparator

未能转换 com.ms.util.IncludeExcludeIntRanges

未能转换 com.ms.util.IncludeExcludeWildcards

未能转换 com.ms.util.ini.IniFile

未能转换 com.ms.util.ini.IniSection

未能转换 com.ms.util.ini.IniSyntaxErrorException

未能转换 com.ms.util.IntRanges.compare

未能转换 com.ms.util.IntRanges.compareSet

未能转换 com.ms.util.IntRanges.ComparisonResulttoString

未能转换 com.ms.util.IntRanges.condense

未能转换 com.ms.util.IntRanges.contains

未能转换 com.ms.util.IntRanges.indexOf

未能转换 com.ms.util.IntRanges.intersect

未能转换 com.ms.util.IntRanges.IntRanges

未能转换 com.ms.util.IntRanges.invertComparisonResult

未能转换 com.ms.util.IntRanges.lock

未能转换 com.ms.util.IntRanges.parse

未能转换 com.ms.util.IntRanges.removeRange

未能转换 com.ms.util.IntRanges.removeRanges

未能转换 com.ms.util.IntRanges.removeRanges(int, int)

未能转换 com.ms.util.IntRanges.removeRanges(int, int, IntRangeComparator)

未能转换 com.ms.util.IntRanges.removeSingleton

未能转换 com.ms.util.IntRanges.size

未能转换 com.ms.util.IntRanges.sort

未能转换 com.ms.util.IntRanges.unlock

未能转换 com.ms.util.IWildcardExpressionComparator

未能转换 com.ms.util.OrdinalMap.insertInCognate

未能转换 com.ms.util.OrdinalMap.moveCognate
未能转换 com.ms.util.OrdinalMap.newCognate
未能转换 com.ms.util.OrdinalMap.removeFromCognate
未能转换 com.ms.util.ProvideSetComparisonInfo
未能转换 com.ms.util.Queue.capacity
未能转换 com.ms.util.Queue.elementFromHead
未能转换 com.ms.util.Queue.elementFromTail
未能转换 com.ms.util.Queue.hasMoreElements
未能转换 com.ms.util.Queue.nextElement
未能转换 com.ms.util.Queue.setCapacity
未能转换 com.ms.util.Queue.toString
未能转换 com.ms.util.SetComparer
未能转换 com.ms.util.SetComparison
未能转换 com.ms.util.SetTemplate.foundAt
未能转换 com.ms.util.SetTemplate.foundCognate
未能转换 com.ms.util.SetTemplate.foundIn
未能转换 com.ms.util.SetTemplate.generation
未能转换 com.ms.util.SetTemplate.insertInCognate
未能转换 com.ms.util.SetTemplate.keys
未能转换 com.ms.util.SetTemplate.locate
未能转换 com.ms.util.SetTemplate.moveCognate
未能转换 com.ms.util.SetTemplate.newCognate
未能转换 com.ms.util.SetTemplate.rehash
未能转换 com.ms.util.SetTemplate.removeFromCognate
未能转换 com.ms.util.SetTemplate.reportFinds
未能转换 com.ms.util.SetTemplate.reportRequests
未能转换 com.ms.util.SetTemplate.reportUnits
未能转换 com.ms.util.SetTemplate.SetTemplate
未能转换 com.ms.util.Sort.compare
未能转换 com.ms.util.Sort.doSort
未能转换 com.ms.util.Sort.swap
未能转换 com.ms.util.StringComparisonascending
未能转换 com.ms.util.StringComparisondescending
未能转换 com.ms.util.SystemVersionManager
未能转换 com.ms.util.Task
未能转换 com.ms.util.TaskManager
未能转换 com.ms.util.ThreadLocalStorage
未能转换 com.ms.util.Timer.getRepeat
未能转换 com.ms.util.Timer.getUserID
未能转换 com.ms.util.Timer.Timer

未能转换 com.ms.util.TimerEvent.getSource
未能转换 com.ms.util.TimerEvent.getTime
未能转换 com.ms.util.TimerEvent.getUserID
未能转换 com.ms.util.TimerListener
未能转换 com.ms.util.UnsignedIntRanges.contains
未能转换 com.ms.util.UnsignedIntRanges.indexOf
未能转换 com.ms.util.UnsignedIntRanges.intersect
未能转换 com.ms.util.UnsignedIntRanges.removeRange
未能转换 com.ms.util.UnsignedIntRanges.UnsignedIntRanges
未能转换 com.ms.util.VectorSort.compare
未能转换 com.ms.util.VectorSort.swap
未能转换 com.ms.util.WildcardExpression
未能转换 com.ms.util.zip.ZipInputStreamEx
未能转换 com.ms.util.zip.ZipOutputStreamEx

Com.ms.wfc 错误信息

未能转换 com.ms.wfc.Rectangle.Editor

Com.ms.wfc.app 错误信息

未能转换 com.ms.wfc.app.Application.addOnSettingChange
未能转换 com.ms.wfc.app.Application.addOnSystemShutdown
未能转换 com.ms.wfc.app.Application.allocThreadStorage
未能转换 com.ms.wfc.app.Application.createThread
未能转换 com.ms.wfc.app.Application.doEvents
未能转换 com.ms.wfc.app.Application.freeThreadStorage
未能转换 com.ms.wfc.app.Application.getParkingForm
未能转换 com.ms.wfc.app.Application.getThreadStorage
未能转换 com.ms.wfc.app.Application.removeOnSettingChange
未能转换 com.ms.wfc.app.Application.removeOnSystemShutdown
未能转换 com.ms.wfc.app.Application.runDialog
未能转换 com.ms.wfc.app.Application.setThreadStorage
未能转换 com.ms.wfc.app.CharacterSet
未能转换 com.ms.wfc.app.Clipboard.Clipboard
未能转换 com.ms.wfc.app.DataFormats.CF_CSV
未能转换 com.ms.wfc.app.DataFormats.CF_WFCOBJECT
未能转换 com.ms.wfc.app.DataFormats.DataFormats
未能转换 com.ms.wfc.app.DataFormats.Format.Format
未能转换 com.ms.wfc.app.DataFormats.Format.win32Handle
未能转换 com.ms.wfc.app.DataObject.OleDAdvise
未能转换 com.ms.wfc.app.DataObject.OleDUnadvise
未能转换 com.ms.wfc.app.DataObject.OleEnumDAdvise
未能转换 com.ms.wfc.app.DataObject.OleEnumFormatEtc
未能转换 com.ms.wfc.app.DataObject.OleGetCanonicalFormatEtc
未能转换 com.ms.wfc.app.DataObject.OleGetData
未能转换 com.ms.wfc.app.DataObject.OleGetDataHere
未能转换 com.ms.wfc.app.DataObject.OleQueryGetData
未能转换 com.ms.wfc.app.DataObject.OleSetData
未能转换 com.ms.wfc.app.IMessageFilter.postFilterMessage
未能转换 com.ms.wfc.app.KeywordVk
未能转换 com.ms.wfc.app.Languages.Languages
未能转换 com.ms.wfc.app.Locale.CalendarType
未能转换 com.ms.wfc.app.Locale.compareStrings
未能转换 com.ms.wfc.app.Locale.DateFormatOrder
未能转换 com.ms.wfc.app.Locale.getCalendarType
未能转换 com.ms.wfc.app.Locale.getCenturyFormat
未能转换 com.ms.wfc.app.Locale.getCharacterSet

未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCase
未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCaseKana
未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCaseKashida
未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCaseNonSpace
未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCaseSymbols
未能转换 com.ms.wfc.app.Locale.getCompareIgnoreCaseWidth
未能转换 com.ms.wfc.app.Locale.getCountryCode
未能转换 com.ms.wfc.app.Locale.getDefaultCountry
未能转换 com.ms.wfc.app.Locale.getEnglishCurrencyName
未能转换 com.ms.wfc.app.Locale.getLeadingZero
未能转换 com.ms.wfc.app.Locale.getNativeDigits
未能转换 com.ms.wfc.app.Locale.getSortId
未能转换 com.ms.wfc.app.Locale.getSupportedLocales
未能转换 com.ms.wfc.app.Locale.Languages.Locale.Languages
未能转换 com.ms.wfc.app.Locale.Languages.RHAETO_ROMAN
未能转换 com.ms.wfc.app.Locale.LeadingZeros
未能转换 com.ms.wfc.app.Locale.Locale
未能转换 com.ms.wfc.app.Locale.MeasurementSystem
未能转换 com.ms.wfc.app.Locale.NegativeNumberMode.Locale.NegativeNumberMode
未能转换 com.ms.wfc.app.Locale.OptionalCalendarType
未能转换 com.ms.wfc.app.Locale.PositiveCurrencyMode.Locale.PositiveCurrencyMode
未能转换 com.ms.wfc.app.Locale.setCalendarType
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCase
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCaseKana
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCaseKashida
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCaseNonSpace
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCaseSymbols
未能转换 com.ms.wfc.app.Locale.setCompareIgnoreCaseWidth
未能转换 com.ms.wfc.app.Locale.setFirstDayOfWeek
未能转换 com.ms.wfc.app.Locale.setFirstWeekOfYear
未能转换 com.ms.wfc.app.Locale.setLeadingZero
未能转换 com.ms.wfc.app.Locale.setListSeparator
未能转换 com.ms.wfc.app.Locale.setMeasurementSystem
未能转换 com.ms.wfc.app.Locale.Sort
未能转换 com.ms.wfc.app.Locale.SubLanguages.CHINESE_SIMPLIFIED
未能转换 com.ms.wfc.app.Locale.SubLanguages.CHINESE_TRADITIONAL
未能转换 com.ms.wfc.app.Locale.SubLanguages.DUTCH_BELGIAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_AUS
未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_CAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_EIRE

未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_NZ
未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_UK
未能转换 com.ms.wfc.app.Locale.SubLanguages.ENGLISH_US
未能转换 com.ms.wfc.app.Locale.SubLanguages.FRENCH_BELGIAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.FRENCH_CANADIAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.FRENCH_SWISS
未能转换 com.ms.wfc.app.Locale.SubLanguages.GERMAN_AUSTRIAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.GERMAN_SWISS
未能转换 com.ms.wfc.app.Locale.SubLanguages.ITALIAN_SWISS
未能转换 com.ms.wfc.app.Locale.SubLanguages.Locale.SubLanguages
未能转换 com.ms.wfc.app.Locale.SubLanguages.NORWEGIAN_BOKMAL
未能转换 com.ms.wfc.app.Locale.SubLanguages.NORWEGIAN_NYNORSK
未能转换 com.ms.wfc.app.Locale.SubLanguages.PORTUGUESE_BRAZILIAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.SERBO_CROATIAN_CYRILLIC
未能转换 com.ms.wfc.app.Locale.SubLanguages.SERBO_CROATIAN_LATIN
未能转换 com.ms.wfc.app.Locale.SubLanguages.SPANISH_MEXICAN
未能转换 com.ms.wfc.app.Locale.SubLanguages.SPANISH_MODERN
未能转换 com.ms.wfc.app.Locale.SubLanguages.SubLanguages
未能转换 com.ms.wfc.app.Locale.SubLanguages.SYS_DEFAULT
未能转换 com.ms.wfc.app.Message.free
未能转换 com.ms.wfc.app.MethodInvoker.MethodInvoker
未能转换 com.ms.wfc.app.NegativeNumberMode.NegativeNumberMode
未能转换 com.ms.wfc.app.PositiveCurrencyMode.PositiveCurrencyMode
未能转换 com.ms.wfc.app.Registry.Registry
未能转换 com.ms.wfc.app.RegistryKey.getBaseKey
未能转换 com.ms.wfc.app.SendKeysHookProc
未能转换 com.ms.wfc.app.SKEEvent
未能转换 com.ms.wfc.app.SubLanguages.SubLanguages
未能转换 com.ms.wfc.app.SystemInformation.getArrange
未能转换 com.ms.wfc.app.ThreadExceptionEventHandler.ThreadExceptionEventHandler
com.ms.wfc.app.Time.getConstructorArgs
未能转换 com.ms.wfc.app.TimegetExtension
未能转换 com.ms.wfc.app.Time.save
未能转换 com.ms.wfc.app.Time.Time
未能转换 com.ms.wfc.app.Time.toSystemTime
未能转换 com.ms.wfc.app.Time.toVariant
未能转换 com.ms.wfc.app.Window.callback

Com.ms.wfc.ax 错误信息

未能转换 com.ms.wfc.ax
未能转换 com.ms.wfc.ax._POINTL
未能转换 com.ms.wfc.ax.ActiveX
未能转换 com.ms.wfc.ax.Ambients
未能转换 com.ms.wfc.ax.IAdviseSink
未能转换 com.ms.wfc.ax.ICategorizeProperties
未能转换 com.ms.wfc.ax.IClassFactory
未能转换 com.ms.wfc.ax.IClassFactory2
未能转换 com.ms.wfc.ax.IDDispatch
未能转换 com.ms.wfc.ax.IEDispatch
未能转换 com.ms.wfc.ax.IEnumOLEVERB
未能转换 com.ms.wfc.ax.IEnumUnknown
未能转换 com.ms.wfc.ax.IExtender
未能转换 com.ms.wfc.ax.IFont
未能转换 com.ms.wfc.ax.IFontDisp
未能转换 com.ms.wfc.ax.IGetOleObject
未能转换 com.ms.wfc.ax.IGetVBAObject
未能转换 com.ms.wfc.ax.IMyDispatch
未能转换 com.ms.wfc.ax.IObjectIdentity
未能转换 com.ms.wfc.ax.IObjectWithSite
未能转换 com.ms.wfc.ax.IOleClientSite
未能转换 com.ms.wfc.ax.IOleContainer
未能转换 com.ms.wfc.ax.IOleControl
未能转换 com.ms.wfc.ax.IOleControlSite
未能转换 com.ms.wfc.ax.IOleInPlaceActiveObject
未能转换 com.ms.wfc.ax.IOleInPlaceFrame
未能转换 com.ms.wfc.ax.IOleInPlaceObject
未能转换 com.ms.wfc.ax.IOleInPlaceSite
未能转换 com.ms.wfc.ax.IOleInPlaceUIWindow
未能转换 com.ms.wfc.ax.IOleWindow
未能转换 com.ms.wfc.ax.IParseDisplayName
未能转换 com.ms.wfc.ax.IPerPropertyBrowsing
未能转换 com.ms.wfc.ax.IPersist
未能转换 com.ms.wfc.ax.IPersist.iid
未能转换 com.ms.wfc.ax.IPersistPropertyBag
未能转换 com.ms.wfc.ax.IPersistStorage
未能转换 com.ms.wfc.ax.IPersistStream

未能转换 com.ms.wfc.ax.IPersistStreamInit
未能转换 com.ms.wfc.ax.IPicture
未能转换 com.ms.wfc.ax.IPictureDisp
未能转换 com.ms.wfc.ax.IPropertyBag
未能转换 com.ms.wfc.ax.IQuickActivate
未能转换 com.ms.wfc.ax.ISimpleFrameSite
未能转换 com.ms.wfc.ax.ISpecifyPropertyPages
未能转换 com.ms.wfc.ax.IVBFormat
未能转换 com.ms.wfc.ax.IVBGetControl
未能转换 com.ms.wfc.ax.tagCADWORD
未能转换 com.ms.wfc.ax.tagCALPOLESTR
未能转换 com.ms.wfc.ax.tagCAUUID
未能转换 com.ms.wfc.ax.tagCONTROLINFO
未能转换 com.ms.wfc.ax.tagDISPPARAMS
未能转换 com.ms.wfc.ax.tagLICINFO
未能转换 com.ms.wfc.ax.tagLOGPALETTE
未能转换 com.ms.wfc.ax.tagMSG
未能转换 com.ms.wfc.ax.tagOIFI
未能转换 com.ms.wfc.ax.tagOLEMenuGroupWidths
未能转换 com.ms.wfc.ax.tagOLEVERB
未能转换 com.ms.wfc.ax.tagPOINTF
未能转换 com.ms.wfc.ax.tagQACONTAINER
未能转换 com.ms.wfc.ax.tagQACONTROL
未能转换 com.ms.wfc.ax.tagRECT
未能转换 com.ms.wfc.ax.tagSIZE
未能转换 com.ms.wfc.ax.tagSIZEL

Com.ms.wfc.core 错误信息

未能转换 com.ms.wfc.core.ArrayDialog
未能转换 com.ms.wfc.core.ArrayEditor.createNewInstance
未能转换 com.ms.wfc.core.ArrayEditor.defaultPropInfo
未能转换 com.ms.wfc.core.ArrayEditor.editValue
未能转换 com.ms.wfc.core.ArrayEditor.getBaseName
未能转换 com.ms.wfc.core.ArrayEditor.getDisplayText
未能转换 com.ms.wfc.core.ArrayEditor.getTextFromValue
未能转换 com.ms.wfc.core.ArrayEditor.getTypeDescription
未能转换 com.ms.wfc.core.BooleanEditor.getStyle
未能转换 com.ms.wfc.core.BooleanEditor.getValueFromText
未能转换 com.ms.wfc.core.CancelEventHandler.CancelEventHandler
未能转换 com.ms.wfc.core.CategoryAttribute.Position
未能转换 com.ms.wfc.core.Component.appendEventHandlers
未能转换 com.ms.wfc.core.Component.componentChanged
未能转换 com.ms.wfc.core.Component.getDisposing
未能转换 com.ms.wfc.core.Component.getResource
未能转换 com.ms.wfc.core.Component.getService
未能转换 com.ms.wfc.core.ComponentInfo.getClassInfo
未能转换 com.ms.wfc.core.ComponentInfo.getDefaultEventInfo
未能转换 com.ms.wfc.core.ComponentInfo.getExtenders
未能转换 com.ms.wfc.core.ComponentManager.createClassInfo
未能转换 com.ms.wfc.core.ComponentManager.createValueEditor
未能转换 com.ms.wfc.core.ComponentManager.getComponentInfo
未能转换 com.ms.wfc.core.ComponentManager.getValueEditor
未能转换 com.ms.wfc.core.ComponentManager.registerClassInfo
未能转换 com.ms.wfc.core.ComponentManager.registerValueEditorClass
未能转换 com.ms.wfc.core.ConstructorArg
未能转换 com.ms.wfc.core.ConstructorArg.ConstructorArg
未能转换 com.ms.wfc.core.CustomizerVerb.addOnVerbExecute
未能转换 com.ms.wfc.core.CustomizerVerb.CustomizerVerb
未能转换 com.ms.wfc.core.CustomizerVerb.getBitmap
未能转换 com.ms.wfc.core.CustomizerVerb.getData
未能转换 com.ms.wfc.core.CustomizerVerb.onVerbExecute
未能转换 com.ms.wfc.core.CustomizerVerb.performVerbExecute
未能转换 com.ms.wfc.core.CustomizerVerb.removeOnVerbExecute
未能转换 com.ms.wfc.core.CustomizerVerb.setBitmap
未能转换 com.ms.wfc.core.CustomizerVerb.setData

未能转换 com.ms.wfc.core.DescriptionAttribute.getDescription
未能转换 com.ms.wfc.core.DesignForm.DesignForm
未能转换 com.ms.wfc.core.DesignForm.getPage
未能转换 com.ms.wfc.core.DesignForm.getPageCount
未能转换 com.ms.wfc.core.DesignForm.showForm
未能转换 com.ms.wfc.core.DesignPage.getPageMessage
未能转换 com.ms.wfc.core.DesignPage.NULLVALUE
未能转换 com.ms.wfc.core.DesignPage.objects
未能转换 com.ms.wfc.core.DesignPage.onReadProperty
未能转换 com.ms.wfc.core.DesignPage.onWriteProperty
未能转换 com.ms.wfc.core.DesignPage.properties
未能转换 com.ms.wfc.core.DesignPage.setObjects
未能转换 com.ms.wfc.core.DoubleEditor.getValueFromText
未能转换 com.ms.wfc.core.Enum.Enum
未能转换 com.ms.wfc.core.Event.Event
未能转换 com.ms.wfc.core.Event.extendedInfo
未能转换 com.ms.wfc.core.EventHandler.EventHandler
未能转换 com.ms.wfc.core.EventInfo.addHandler
未能转换 com.ms.wfc.core.EventInfo.getAddMethod
未能转换 com.ms.wfc.core.EventInfo.getMulticast
未能转换 com.ms.wfc.core.EventInfo.getRemoveMethod
未能转换 com.ms.wfc.core.EventInfo.getType
未能转换 com.ms.wfc.core.EventInfo.removeEventHandler
未能转换 com.ms.wfc.core.ExtenderInfo
未能转换 com.ms.wfc.core.FieldsEditor
未能转换 com.ms.wfc.core.FloatEditor.getValueFromText
未能转换 com.ms.wfc.core.IAttributes
未能转换 com.ms.wfc.core.IClassInfo
未能转换 com.ms.wfc.core.IContainer.dispose
未能转换 com.ms.wfc.core.IContainer.getComponent
未能转换 com.ms.wfc.core.IContainer.getComponents
未能转换 com.ms.wfc.core.IContainer.getDesignPages
未能转换 com.ms.wfc.core.IContainer.getHitTest
未能转换 com.ms.wfc.core.IContainer.getVerbs
未能转换 com.ms.wfc.core.IDesignPage.getSize

未能转换 com.ms.wfc.core.IDesignPage.setObjects
未能转换 com.ms.wfc.core.IDesignPage.setSize
未能转换 com.ms.wfc.core.IEditorHost.getHostHandle
未能转换 com.ms.wfc.core.IEditorSite.getType
未能转换 com.ms.wfc.core.IEditorSite.getValueOwner
未能转换 com.ms.wfc.core.IEditorSite.getValueOwners
未能转换 com.ms.wfc.core.IResourceLoader
未能转换 com.ms.wfc.core.IResourceManager.containsProperty
未能转换 com.ms.wfc.core.IResourceManager.getNames
未能转换 com.ms.wfc.core.IResourceManager.getProperties
未能转换 com.ms.wfc.core.IResourceManager.saveAllProperties
未能转换 com.ms.wfc.core.IResourceManager.setProperty
未能转换 com.ms.wfc.core.IResourceManager.setResourceSet
未能转换 com.ms.wfc.core.IValueEditor.getConstantName
未能转换 com.ms.wfc.core.IValueEditor.getValueFromSubPropertyValues
未能转换 com.ms.wfc.core.IValueEditor.STYLE_IMMEDIATE
未能转换 com.ms.wfc.core.IValueEditor.STYLE_NOARRAYEXPANSION
未能转换 com.ms.wfc.core.IValueEditor.STYLE_NOARRAYMULTISELECT
未能转换 com.ms.wfc.core.IValueEditor.STYLE_NOEDITABLETEXT
未能转换 com.ms.wfc.core.IValueEditor.STYLE_PAINTVALUE
未能转换 com.ms.wfc.core.IValueEditor.STYLE_PROPERTIES
未能转换 com.ms.wfc.core.IValueEditor.STYLE_SHOWFIELDS
未能转换 com.ms.wfc.core.IValueEditor.STYLE_VALUES
未能转换 com.ms.wfc.core.MemberAttribute.MemberAttribute
未能转换 com.ms.wfc.core.NonEditableReferenceEditor.getStyle
未能转换 com.ms.wfc.core.NonEditableReferenceEditor.NonEditableReferenceEditor
未能转换 com.ms.wfc.core.NonPersistableAttribute
未能转换 com.ms.wfc.core.PropertyInfo.canResetValue
未能转换 com.ms.wfc.core.PropertyInfo.getGetMethod
未能转换 com.ms.wfc.core.PropertyInfo.getIsReadOnly
未能转换 com.ms.wfc.core.PropertyInfo.getResetMethod
未能转换 com.ms.wfc.core.PropertyInfo.getSetMethod
未能转换 com.ms.wfc.core.PropertyInfo.getShouldPersistMethod
未能转换 com.ms.wfc.core.PropertyInfo.getType
未能转换 com.ms.wfc.core.PropertyInfo.getValue
未能转换 com.ms.wfc.core.PropertyInfo.getValueEditor
未能转换 com.ms.wfc.core.PropertyInfo.resetValue
未能转换 com.ms.wfc.core.PropertyInfo.setValue
未能转换 com.ms.wfc.core.PropertyInfo.shouldPersistValue
未能转换 com.ms.wfc.core.ReferenceEditor.container

未能转换 com.ms.wfc.core.ReferenceEditor.getStyle
未能转换 com.ms.wfc.core.ReferenceEditor.getTextFromValue
未能转换 com.ms.wfc.core.ReferenceEditor.getValueFromText
未能转换 com.ms.wfc.core.ReferenceEditor.getValues
未能转换 com.ms.wfc.core.ReferenceEditor.none
未能转换 com.ms.wfc.core.ReferenceEditor.ReferenceEditor
未能转换 com.ms.wfc.core.ReferenceEditor.type
未能转换 com.ms.wfc.core.ResourceManager.containsProperty
未能转换 com.ms.wfc.core.ResourceManager.EXTENSION
未能转换 com.ms.wfc.core.ResourceManager.getLocaleFileNameFromBase
未能转换 com.ms.wfc.core.ResourceManager.getNames
未能转换 com.ms.wfc.core.ResourceManager.getProperties
未能转换 com.ms.wfc.core.ResourceManager.getResource
未能转换 com.ms.wfc.core.ResourceManager.getResourceLoader
未能转换 com.ms.wfc.core.ResourceManager.ResourceManager
未能转换 com.ms.wfc.core.ResourceManager.save
未能转换 com.ms.wfc.core.ResourceManager.saveAllProperties
未能转换 com.ms.wfc.core.ResourceManager.setBaseName
未能转换 com.ms.wfc.core.ResourceManager.setProperty
未能转换 com.ms.wfc.core.ResourceManager.setResourceLoader
未能转换 com.ms.wfc.core.ResourceManager.setResourceSet
未能转换 com.ms.wfc.core.ResourceReader
未能转换 com.ms.wfc.core.ResourceWriter
未能转换 com.ms.wfc.core.ShowInToolboxAttribute.NO
未能转换 com.ms.wfc.core.ShowInToolboxAttribute.YES
未能转换 com.ms.wfc.core.StringListDialog
未能转换 com.ms.wfc.core.StringListEditor
未能转换 com.ms.wfc.core.Sys
未能转换 com.ms.wfc.core.ValueEditor.editValue
未能转换 com.ms.wfc.core.ValueEditor.getValueFromSubPropertyValues
未能转换 com.ms.wfc.core.ValueEditorAttribute.valueEditorType
未能转换 com.ms.wfc.core.VerbExecuteEvent
未能转换 com.ms.wfc.core.VerbExecuteEventHandler
未能转换 com.ms.wfc.core.WFCEception.getBaseException
未能转换 com.ms.wfc.core.WFCEception.printStackTrace

Com.ms.wfc.data 错误信息

未能转换 com.ms.wfc.data.AdoEnums
未能转换 com.ms.wfc.data.AdoEvent
未能转换 com.ms.wfc.data.AdoException.AdoException
未能转换 com.ms.wfc.data.AdoProperties.AdoProperties
未能转换 com.ms.wfc.data.BooleanDataFormat
未能转换 com.ms.wfc.data.Connection.addOnBeginTransComplete
未能转换 com.ms.wfc.data.Connection.addOnCommitTransComplete
未能转换 com.ms.wfc.data.Connection.addOnConnectComplete
未能转换 com.ms.wfc.data.Connection.addOnDisconnect
未能转换 com.ms.wfc.data.Connection.addOnExecuteComplete
未能转换 com.ms.wfc.data.Connection.addOnInfoMessage
未能转换 com.ms.wfc.data.Connection.addOnRollbackTransComplete
未能转换 com.ms.wfc.data.Connection.addOnWillConnect
未能转换 com.ms.wfc.data.Connection.addOnWillExecute
未能转换 com.ms.wfc.data.Connection.Connection
未能转换 com.ms.wfc.data.Connection.getPeer
未能转换 com.ms.wfc.data.Connection.openSchema
未能转换 com.ms.wfc.data.Connection.removeOnBeginTransComplete
未能转换 com.ms.wfc.data.Connection.removeOnCommitTransComplete
未能转换 com.ms.wfc.data.Connection.removeOnConnectComplete
未能转换 com.ms.wfc.data.Connection.removeOnDisconnect
未能转换 com.ms.wfc.data.Connection.removeOnExecuteComplete
未能转换 com.ms.wfc.data.Connection.removeOnInfoMessage
未能转换 com.ms.wfc.data.Connection.removeOnRollbackTransComplete
未能转换 com.ms.wfc.data.Connection.removeOnWillConnect
未能转换 com.ms.wfc.data.Connection.removeOnWillExecute
未能转换 com.ms.wfc.data.Connection.setConnectionString
未能转换 com.ms.wfc.data.ConnectionEvent
未能转换 com.ms.wfc.data.ConnectionEventHandler
未能转换 com.ms.wfc.data.DataFormat
未能转换 com.ms.wfc.data.DataFormat.FormatWrapper
未能转换 com.ms.wfc.data.DateFormat.DateFormat
未能转换 com.ms.wfc.data.DateFormat.valid
未能转换 com.ms.wfc.data.dsl.<ClassName>.clsid
未能转换 com.ms.wfc.data.dsl.<ClassName>.iid
未能转换 com.ms.wfc.data.dsl._COAUTHIDENTITY._COAUTHIDENTITY
未能转换 com.ms.wfc.data.dsl._COAUTHINFO._COAUTHINFO

未能转换 com.ms.wfc.data.dsl._COSERVERINFO._COSERVERINFO
未能转换 com.ms.wfc.data.dsl._RemotableHandle._RemotableHandle
未能转换 com.ms.wfc.data.dsl.tagMULTI_QI
未能转换 com.ms.wfc.data.EventHandlerList.clear
未能转换 com.ms.wfc.data.EventHandlerList.getList
未能转换 com.ms.wfc.data.EventsListener
未能转换 com.ms.wfc.data.Field.getDataTimestamp
未能转换 com.ms.wfc.data.Field.getDispatch
未能转换 com.ms.wfc.data.Field.getGuid
未能转换 com.ms.wfc.data.Field.getInt
未能转换 com.ms.wfc.data.Field.getObject
未能转换 com.ms.wfc.data.Field.getTime
未能转换 com.ms.wfc.data.Field.getTimestamp
未能转换 com.ms.wfc.data.Field.setDataDate
未能转换 com.ms.wfc.data.Field.setDispatch
未能转换 com.ms.wfc.data.Field.setGuid
未能转换 com.ms.wfc.data.Field.setTime
未能转换 com.ms.wfc.data.Field.setTimestamp
未能转换 com.ms.wfc.data.IDataFormat
未能转换 com.ms.wfc.data.IDataFormat.Editor
未能转换 com.ms.wfc.data.IDataSource.addDataSourceListener
未能转换 com.ms.wfc.data.IDataSource.getDataMember
未能转换 com.ms.wfc.data.IDataSource.getDataMemberCount
未能转换 com.ms.wfc.data.IDataSource.getDataMemberName
未能转换 com.ms.wfc.data.IDataSource.iid
未能转换 com.ms.wfc.data.IDataSource.removeDataSourceListener
未能转换 com.ms.wfc.data.IDataSourceListener
未能转换 com.ms.wfc.data.IMarshal.iid
未能转换 com.ms.wfc.data.IPersist.iid
未能转换 com.ms.wfc.data.IPersistStream.GetSizeMax
未能转换 com.ms.wfc.data.IPersistStream.iid
未能转换 com.ms.wfc.data.IPersistStream.Load
未能转换 com.ms.wfc.data.NumberDataFormat
未能转换 com.ms.wfc.data.ObjectProxy.call
未能转换 com.ms.wfc.data.rds.<ClassName>.clsid
未能转换 com.ms.wfc.data.rds.<ClassName>.iid
未能转换 com.ms.wfc.data.Recordset.addDataSourceListener
未能转换 com.ms.wfc.data.Recordset.addNew
未能转换 com.ms.wfc.data.Recordset.addOnEndOfRecordset
未能转换 com.ms.wfc.data.Recordset.addOnFetchComplete

未能转换 com.ms.wfc.data.Recordset.addOnFetchProgress
未能转换 com.ms.wfc.data.Recordset.addOnFieldChangeComplete
未能转换 com.ms.wfc.data.Recordset.addOnMoveComplete
未能转换 com.ms.wfc.data.Recordset.addOnRecordChangeComplete
未能转换 com.ms.wfc.data.Recordset.addOnRecordsetChangeComplete
未能转换 com.ms.wfc.data.Recordset.addOnWillChangeField
未能转换 com.ms.wfc.data.Recordset.addOnWillChangeRecord
未能转换 com.ms.wfc.data.Recordset.addOnWillChangeRecordset
未能转换 com.ms.wfc.data.Recordset.addOnWillMove
未能转换 com.ms.wfc.data.Recordset.DisconnectObject
未能转换 com.ms.wfc.data.Recordset.GetClassID
未能转换 com.ms.wfc.data.Recordset.getCommand
未能转换 com.ms.wfc.data.Recordset.getDataMember
未能转换 com.ms.wfc.data.Recordset.getDataMemberCount
未能转换 com.ms.wfc.data.Recordset.getDataMemberName
未能转换 com.ms.wfc.data.Recordset.GetMarshalSizeMax
未能转换 com.ms.wfc.data.Recordset.getRecordset
未能转换 com.ms.wfc.data.Recordset.getRows
未能转换 com.ms.wfc.data.Recordset.GetSizeMax
未能转换 com.ms.wfc.data.Recordset.GetUnmarshalClass
未能转换 com.ms.wfc.data.Recordset.IsDirty
未能转换 com.ms.wfc.data.Recordset.Load
未能转换 com.ms.wfc.data.Recordset.MarshallInterface
未能转换 com.ms.wfc.data.Recordset.move
未能转换 com.ms.wfc.data.Recordset.nextRecordset
未能转换 com.ms.wfc.data.Recordset.open
未能转换 com.ms.wfc.data.Recordset.open(Object)
未能转换 com.ms.wfc.data.Recordset.Recordset(IDataSource)
未能转换 com.ms.wfc.data.Recordset.Recordset(IDataSource, String)
未能转换 com.ms.wfc.data.Recordset.Recordset(Object)
未能转换 com.ms.wfc.data.Recordset.release
未能转换 com.ms.wfc.data.Recordset.ReleaseMarshalData
未能转换 com.ms.wfc.data.Recordset.removeDataSourceListener
未能转换 com.ms.wfc.data.Recordset.removeOnEndOfRecordset
未能转换 com.ms.wfc.data.Recordset.removeOnFetchComplete
未能转换 com.ms.wfc.data.Recordset.removeOnFetchProgress
未能转换 com.ms.wfc.data.Recordset.removeOnFieldChangeComplete
未能转换 com.ms.wfc.data.Recordset.removeOnMoveComplete
未能转换 com.ms.wfc.data.Recordset.removeOnRecordChangeComplete
未能转换 com.ms.wfc.data.Recordset.removeOnRecordsetChangeComplete

未能转换 com.ms.wfc.data.Recordset.removeOnWillChangeField
未能转换 com.ms.wfc.data.Recordset.removeOnWillChangeRecord
未能转换 com.ms.wfc.data.Recordset.removeOnWillChangeRecordset
未能转换 com.ms.wfc.data.Recordset.removeOnWillMove
未能转换 com.ms.wfc.data.Recordset.setCommand
未能转换 com.ms.wfc.data.Recordset.UnmarshallInterface
未能转换 com.ms.wfc.data.RecordsetEvent
未能转换 com.ms.wfc.data.RecordsetEventHandler
未能转换 com.ms.wfc.data.StringManager.getString

Com.ms.wfc.data.adodb 错误信息

未能转换 com.ms.wfc.data.adodb.<ClassName>.clsid

未能转换 com.ms.wfc.data.adodb.<ClassName>.iid

未能转换 com.ms.wfc.data.adodb._Command.setActiveConnection

未能转换 com.ms.wfc.data.adodb._Connection.setConnectionString

未能转换 com.ms.wfc.data.adodb._Recordset.getCollect

未能转换 com.ms.wfc.data.adodb._Recordset.setCollect

未能转换 com.ms.wfc.data.adodb.Command.setActiveConnection

未能转换 com.ms.wfc.data.adodb.Connection.setConnectionString

未能转换 com.ms.wfc.data.adodb.Recordset.getCollect

未能转换 com.ms.wfc.data.adodb.Recordset.setCollect

Com.ms.wfc.data.ui 错误信息

未能转换 com.ms.wfc.data.ui.Column.getAllowSizing
未能转换 com.ms.wfc.data.ui.Column.getBackColor
未能转换 com.ms.wfc.data.ui.Column.getDataFormat
未能转换 com.ms.wfc.data.ui.Column.getDataType
未能转换 com.ms.wfc.data.ui.Column.getFont
未能转换 com.ms.wfc.data.ui.Column.getForeColor
未能转换 com.ms.wfc.data.ui.Column.getSelectedBackColor
未能转换 com.ms.wfc.data.ui.Column.getSelectedForeColor
未能转换 com.ms.wfc.data.ui.Column.getVisible
未能转换 com.ms.wfc.data.ui.Column.getWriteAllowed
未能转换 com.ms.wfc.data.ui.Column.setAlignment
未能转换 com.ms.wfc.data.ui.Column.setAllowSizing
未能转换 com.ms.wfc.data.ui.Column.setBackColor
未能转换 com.ms.wfc.data.ui.Column.setDataFormat
未能转换 com.ms.wfc.data.ui.Column.setFont
未能转换 com.ms.wfc.data.ui.Column.setForeground
未能转换 com.ms.wfc.data.ui.Column.setIndex
未能转换 com.ms.wfc.data.ui.Column.setSelectedBackColor
未能转换 com.ms.wfc.data.ui.Column.setSelectedForeColor
未能转换 com.ms.wfc.data.ui.Column.setValue
未能转换 com.ms.wfc.data.ui.Column.setVisible
未能转换 com.ms.wfc.data.ui.Column.shouldPersistBackColor
未能转换 com.ms.wfc.data.ui.Column.shouldPersistFont
未能转换 com.ms.wfc.data.ui.Column.shouldPersistForeColor
未能转换 com.ms.wfc.data.ui.Column.shouldPersistSelectedBackColor
未能转换 com.ms.wfc.data.ui.Column.shouldPersistSelectedForeColor
未能转换 com.ms.wfc.data.ui.ColumnEditingEvent
未能转换 com.ms.wfc.data.ui.ColumnEditingEventHandler
未能转换 com.ms.wfc.data.ui.ColumnEvent
未能转换 com.ms.wfc.data.ui.ColumnEventHandler
未能转换 com.ms.wfc.data.ui.ColumnResizeEvent
未能转换 com.ms.wfc.data.ui.ColumnResizeEventHandler
未能转换 com.ms.wfc.data.ui.ColumnsEditor
未能转换 com.ms.wfc.data.ui.ColumnsEditorDialog
未能转换 com.ms.wfc.data.ui.ColumnUpdatingEvent
未能转换 com.ms.wfc.data.ui.ColumnUpdatingEventHandler
未能转换 com.ms.wfc.data.ui.ConnectionStringEditor

未能转换 com.ms.wfc.data.ui.DataBinder
未能转换 com.ms.wfc.data.ui.DataBinder.Customizer
未能转换 com.ms.wfc.data.ui.DataBinder.GeneralPage
未能转换 com.ms.wfc.data.ui.DataBinding.bindTarget
未能转换 com.ms.wfc.data.ui.DataBinding.CallbackType
未能转换 com.ms.wfc.data.ui.DataBinding.commitChange
未能转换 com.ms.wfc.data.ui.DataBinding.DataBinding
未能转换 com.ms.wfc.data.ui.DataBinding.FalseArgument
未能转换 com.ms.wfc.data.ui.DataBinding.getConstructorArgs
未能转换 com.ms.wfc.data.ui.DataBinding.getDataFormat
未能转换 com.ms.wfc.data.ui.DataBinding.getFieldName
未能转换 com.ms.wfc.data.ui.DataBinding.getPropertyValue
未能转换 com.ms.wfc.data.ui.DataBinding.getTarget
未能转换 com.ms.wfc.data.ui.DataBinding.m_propertyChange
未能转换 com.ms.wfc.data.ui.DataBinding.onChanged
未能转换 com.ms.wfc.data.ui.DataBinding.onChanging
未能转换 com.ms.wfc.data.ui.DataBinding.onTargetDispose
未能转换 com.ms.wfc.data.ui.DataBinding.propertyChange
未能转换 com.ms.wfc.data.ui.DataBinding.refreshPropertyValue
未能转换 com.ms.wfc.data.ui.DataBinding.setDataFormat
未能转换 com.ms.wfc.data.ui.DataBinding.setFieldName
未能转换 com.ms.wfc.data.ui.DataBinding.setPropertyValue
未能转换 com.ms.wfc.data.ui.DataBinding.setTarget
未能转换 com.ms.wfc.data.ui.DataGrid.addColumn
未能转换 com.ms.wfc.data.ui.DataGrid.addOnColumnEdited
未能转换 com.ms.wfc.data.ui.DataGrid.addOnColumnEditing
未能转换 com.ms.wfc.data.ui.DataGrid.addOnColumnResize
未能转换 com.ms.wfc.data.ui.DataGrid.addOnColumnUpdated
未能转换 com.ms.wfc.data.ui.DataGrid.addOnColumnUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.addOnDeleted
未能转换 com.ms.wfc.data.ui.DataGrid.addOnDeleting
未能转换 com.ms.wfc.data.ui.DataGrid.addOnError
未能转换 com.ms.wfc.data.ui.DataGrid.addOnHeaderClick
未能转换 com.ms.wfc.data.ui.DataGrid.addOnInserted
未能转换 com.ms.wfc.data.ui.DataGrid.addOnInserting
未能转换 com.ms.wfc.data.ui.DataGrid.addOnPositionChange
未能转换 com.ms.wfc.data.ui.DataGrid.addOnRowResize
未能转换 com.ms.wfc.data.ui.DataGrid.addOnScroll
未能转换 com.ms.wfc.data.ui.DataGrid.addOnSelChange
未能转换 com.ms.wfc.data.ui.DataGrid.addOnUpdated

未能转换 com.ms.wfc.data.ui.DataGrid.addOnUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.Customizer
未能转换 com.ms.wfc.data.ui.DataGrid.DataGrid
未能转换 com.ms.wfc.data.ui.DataGrid.DataGridErrorDialog
未能转换 com.ms.wfc.data.ui.DataGrid.fireErrorEvent
未能转换 com.ms.wfc.data.ui.DataGrid.getAllowAddNew
未能转换 com.ms.wfc.data.ui.DataGrid.getAllowArrows
未能转换 com.ms.wfc.data.ui.DataGrid.getAllowDelete
未能转换 com.ms.wfc.data.ui.DataGrid.getAllowRowSizing
未能转换 com.ms.wfc.data.ui.DataGrid.getAllowUpdate
未能转换 com.ms.wfc.data.ui.DataGrid.getCurrentRow
未能转换 com.ms.wfc.data.ui.DataGrid.getCurrentRowModified
未能转换 com.ms.wfc.data.ui.DataGrid.getDisplayIndex
未能转换 com.ms.wfc.data.ui.DataGrid.getDynamicColumns
未能转换 com.ms.wfc.data.ui.DataGrid.getEnterAction
未能转换 com.ms.wfc.data.ui.DataGrid.getFirstRow
未能转换 com.ms.wfc.data.ui.DataGrid.getHeaderLineCount
未能转换 com.ms.wfc.data.ui.DataGrid.getRowHeight
未能转换 com.ms.wfc.data.ui.DataGrid.getScrollbars
未能转换 com.ms.wfc.data.ui.DataGrid.getSelectedColumns
未能转换 com.ms.wfc.data.ui.DataGrid.getSelectedRows
未能转换 com.ms.wfc.data.ui.DataGrid.getShowPhantom
未能转换 com.ms.wfc.data.ui.DataGrid.getTabAction
未能转换 com.ms.wfc.data.ui.DataGrid.getWrapCellPointer
未能转换 com.ms.wfc.data.ui.DataGrid.makeCurrentCellVisible
未能转换 com.ms.wfc.data.ui.DataGrid.MIN_COLUMN_WIDTH
未能转换 com.ms.wfc.data.ui.DataGrid.MIN_ROW_HEIGHT
未能转换 com.ms.wfc.data.ui.DataGrid.onColumnEdited
未能转换 com.ms.wfc.data.ui.DataGrid.onColumnEditing
未能转换 com.ms.wfc.data.ui.DataGrid.onColumnResize
未能转换 com.ms.wfc.data.ui.DataGrid.onColumnUpdated
未能转换 com.ms.wfc.data.ui.DataGrid.onColumnUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.onDeleted
未能转换 com.ms.wfc.data.ui.DataGrid.onDeleting
未能转换 com.ms.wfc.data.ui.DataGrid.onError
未能转换 com.ms.wfc.data.ui.DataGrid.onHeaderClick
未能转换 com.ms.wfc.data.ui.DataGrid.onInserted
未能转换 com.ms.wfc.data.ui.DataGrid.onInserting
未能转换 com.ms.wfc.data.ui.DataGrid.onPositionChange
未能转换 com.ms.wfc.data.ui.DataGrid.onRowResize

未能转换 com.ms.wfc.data.ui.DataGrid.onScroll
未能转换 com.ms.wfc.data.ui.DataGrid.onSelChange
未能转换 com.ms.wfc.data.ui.DataGrid.onUpdated
未能转换 com.ms.wfc.data.ui.DataGrid.onUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.rebind
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnColumnEdited
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnColumnEditing
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnColumnResize
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnColumnUpdated
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnColumnUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnDeleted
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnDeleting
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnError
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnHeaderClick
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnInserted
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnInserting
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnPositionChange
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnRowResize
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnScroll
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnSelChange
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnUpdated
未能转换 com.ms.wfc.data.ui.DataGrid.removeOnUpdating
未能转换 com.ms.wfc.data.ui.DataGrid.rowBookmark
未能转换 com.ms.wfc.data.ui.DataGrid.rowTop
未能转换 com.ms.wfc.data.ui.DataGrid.scroll
未能转换 com.ms.wfc.data.ui.DataGrid.setAllowAddNew
未能转换 com.ms.wfc.data.ui.DataGrid.setAllowArrows
未能转换 com.ms.wfc.data.ui.DataGrid.setAllowDelete
未能转换 com.ms.wfc.data.ui.DataGrid.setAllowRowSizing
未能转换 com.ms.wfc.data.ui.DataGrid.setAllowUpdate
未能转换 com.ms.wfc.data.ui.DataGrid.setCurrentRow
未能转换 com.ms.wfc.data.ui.DataGrid.setDynamicColumns
未能转换 com.ms.wfc.data.ui.DataGrid.setEnterAction
未能转换 com.ms.wfc.data.ui.DataGrid.setFirstRow
未能转换 com.ms.wfc.data.ui.DataGrid.setHeaderLineCount
未能转换 com.ms.wfc.data.ui.DataGrid.setLeftColumn
未能转换 com.ms.wfc.data.ui.DataGrid.setRowHeight
未能转换 com.ms.wfc.data.ui.DataGrid.setScrollbars
未能转换 com.ms.wfc.data.ui.DataGrid.setSelectedColumns
未能转换 com.ms.wfc.data.ui.DataGrid.setSelectedRows

未能转换 com.ms.wfc.data.ui.DataGrid.setTabAction
未能转换 com.ms.wfc.data.ui.DataGrid.setWrapCellPointer
未能转换 com.ms.wfc.data.ui.DataMemberEditor
未能转换 com.ms.wfc.data.ui.DataNavigator.Customizer
未能转换 com.ms.wfc.data.ui.DataNavigator.getDataMember
未能转换 com.ms.wfc.data.ui.DataNavigator.moveFirst
未能转换 com.ms.wfc.data.ui.DataNavigator.moveLast
未能转换 com.ms.wfc.data.ui.DataNavigator.moveNext
未能转换 com.ms.wfc.data.ui.DataNavigator.movePrevious
未能转换 com.ms.wfc.data.ui.DataNavigator.propertyChanged
未能转换 com.ms.wfc.data.ui.DataNavigator.setDataMember
未能转换 com.ms.wfc.data.ui.DataNavigator.setDataSource
未能转换 com.ms.wfc.data.ui.DataSource.addDataSourceListener
未能转换 com.ms.wfc.data.ui.DataSource.addOnBeginTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnCommitTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnConnectComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnDisconnect
未能转换 com.ms.wfc.data.ui.DataSource.addOnEndOfRecordset
未能转换 com.ms.wfc.data.ui.DataSource.addOnExecuteComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnFetchComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnFetchProgress
未能转换 com.ms.wfc.data.ui.DataSource.addOnFieldChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnInfoMessage
未能转换 com.ms.wfc.data.ui.DataSource.addOnMoveComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnRecordChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnRecordsetChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnRollbackTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillChangeField
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillChangeRecord
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillChangeRecordset
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillConnect
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillExecute
未能转换 com.ms.wfc.data.ui.DataSource.addOnWillMove
未能转换 com.ms.wfc.data.ui.DataSource.dataMemberAdded
未能转换 com.ms.wfc.data.ui.DataSource.dataMemberChanged
未能转换 com.ms.wfc.data.ui.DataSource.dataMemberRemoved
未能转换 com.ms.wfc.data.ui.DataSource.DataSource
未能转换 com.ms.wfc.data.ui.DataSource.getAsyncConnect
未能转换 com.ms.wfc.data.ui.DataSource.getAsyncExecute
未能转换 com.ms.wfc.data.ui.DataSource.getAsyncFetch

未能转换 com.ms.wfc.data.ui.DataSource.getCacheSize
未能转换 com.ms.wfc.data.ui.DataSource.getCursorLocation
未能转换 com.ms.wfc.data.ui.DataSource.getCursorType
未能转换 com.ms.wfc.data.ui.DataSource.getDataMember
未能转换 com.ms.wfc.data.ui.DataSource.getDataMemberCount
未能转换 com.ms.wfc.data.ui.DataSource.getDataMemberName
未能转换 com.ms.wfc.data.ui.DataSource.getDesignTimeData
未能转换 com.ms.wfc.data.ui.DataSource.getIsolationLevel
未能转换 com.ms.wfc.data.ui.DataSource.getLockType
未能转换 com.ms.wfc.data.ui.DataSource.getMaxRecords
未能转换 com.ms.wfc.data.ui.DataSource.getMode
未能转换 com.ms.wfc.data.ui.DataSource.getParentDataSource
未能转换 com.ms.wfc.data.ui.DataSource.getParentFieldName
未能转换 com.ms.wfc.data.ui.DataSource.getPassword
未能转换 com.ms.wfc.data.ui.DataSource.getPrepared
未能转换 com.ms.wfc.data.ui.DataSource.getRecordset
未能转换 com.ms.wfc.data.ui.DataSource.getSort
未能转换 com.ms.wfc.data.ui.DataSource.getStayInSync
未能转换 com.ms.wfc.data.ui.DataSource.getUserId
未能转换 com.ms.wfc.data.ui.DataSource.isChildDataSource
未能转换 com.ms.wfc.data.ui.DataSource.removeDataSourceListener
未能转换 com.ms.wfc.data.ui.DataSource.removeOnBeginTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnCommitTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnConnectComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnDisconnect
未能转换 com.ms.wfc.data.ui.DataSource.removeOnEndOfRecordset
未能转换 com.ms.wfc.data.ui.DataSource.removeOnExecuteComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnFetchComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnFetchProgress
未能转换 com.ms.wfc.data.ui.DataSource.removeOnFieldChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnInfoMessage
未能转换 com.ms.wfc.data.ui.DataSource.removeOnMoveComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnRecordChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnRecordsetChangeComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnRollbackTransComplete
未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillChangeField
未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillChangeRecord
未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillChangeRecordset
未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillConnect
未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillExecute

未能转换 com.ms.wfc.data.ui.DataSource.removeOnWillMove
未能转换 com.ms.wfc.data.ui.DataSource.setAsyncConnect
未能转换 com.ms.wfc.data.ui.DataSource.setAsyncExecute
未能转换 com.ms.wfc.data.ui.DataSource.setAsyncFetch
未能转换 com.ms.wfc.data.ui.DataSource.setCacheSize
未能转换 com.ms.wfc.data.ui.DataSource.setConnectionString
未能转换 com.ms.wfc.data.ui.DataSource.setConnectionTimeout
未能转换 com.ms.wfc.data.ui.DataSource.setCursorLocation
未能转换 com.ms.wfc.data.ui.DataSource.setCursorType
未能转换 com.ms.wfc.data.ui.DataSource.setDesignTimeData
未能转换 com.ms.wfc.data.ui.DataSource.setFilter
未能转换 com.ms.wfc.data.ui.DataSource.setIsolationLevel
未能转换 com.ms.wfc.data.ui.DataSource.setLockType
未能转换 com.ms.wfc.data.ui.DataSource.setMaxRecords
未能转换 com.ms.wfc.data.ui.DataSource.setMode
未能转换 com.ms.wfc.data.ui.DataSource.setParentDataSource
未能转换 com.ms.wfc.data.ui.DataSource.setParentFieldName
未能转换 com.ms.wfc.data.ui.DataSource.setPassword
未能转换 com.ms.wfc.data.ui.DataSource.setPrepared
未能转换 com.ms.wfc.data.ui.DataSource.setSort
未能转换 com.ms.wfc.data.ui.DataSource.setStayInSync
未能转换 com.ms.wfc.data.ui.DataSource.setUserId
未能转换 com.ms.wfc.data.ui.DataSource.unrealize
未能转换 com.ms.wfc.data.ui.EnterAction
未能转换 com.ms.wfc.data.ui.ErrorEvent
未能转换 com.ms.wfc.data.ui.ErrorEventHandler
未能转换 com.ms.wfc.data.ui.FieldNameEditor
未能转换 com.ms.wfc.data.ui.GridLineStyle.GridLineStyle
未能转换 com.ms.wfc.data.ui.GridLineStyle.RAISED3D
未能转换 com.ms.wfc.data.ui.GridLineStyle.SUNKEN3D
未能转换 com.ms.wfc.data.ui.PositionChangeEvent.lastColumn
未能转换 com.ms.wfc.data.ui.PositionChangeEvent.lastRow
未能转换 com.ms.wfc.data.ui.PositionChangeEvent.PositionChangeEvent
未能转换 com.ms.wfc.data.ui.PositionChangeEvent.Handler.PositionChangeEvent.Handler
未能转换 com.ms.wfc.data.ui.PropertyNameEditor
未能转换 com.ms.wfc.data.ui.RecordsetProvider
未能转换 com.ms.wfc.data.ui.TabAction

Com.ms.wfc.io 错误信息

未能转换 com.ms.wfc.io.BufferedStream.getStream
未能转换 com.ms.wfc.io.BufferedStream.readCore
未能转换 com.ms.wfc.io.BufferedStream.readStringCharsAnsi
未能转换 com.ms.wfc.io.BufferedStream.readStringNull
未能转换 com.ms.wfc.io.BufferedStream.readStringNullAnsi
未能转换 com.ms.wfc.io.BufferedStream.writeCore
未能转换 com.ms.wfc.io.CodePage.ANSI
未能转换 com.ms.wfc.io.CodePage.CodePage
未能转换 com.ms.wfc.io.CodePage.MAC
未能转换 com.ms.wfc.io.CodePage.OEM
未能转换 com.ms.wfc.io.DataStream.comStream
未能转换 com.ms.wfc.io.DataStream.DataStream
未能转换 com.ms.wfc.io.DataStream.fromComStream
未能转换 com.ms.wfc.io.DataStream.getComStream
未能转换 com.ms.wfc.io.DataStream.readCore
未能转换 com.ms.wfc.io.DataStream.readStringNull
未能转换 com.ms.wfc.io.DataStream.readStringNullAnsi
未能转换 com.ms.wfc.io.DataStream.readUTF
未能转换 com.ms.wfc.io.DataStream.toComStream
未能转换 com.ms.wfc.io.DataStream.writeCore
未能转换 com.ms.wfc.io.DataStream.writeStringChars
未能转换 com.ms.wfc.io.DataStream.writeStringCharsAnsi
未能转换 com.ms.wfc.io.DataStream.writeStringNull
未能转换 com.ms.wfc.io.DataStream.writeStringNullAnsi
未能转换 com.ms.wfc.io.DataStream.writeUTF
未能转换 com.ms.wfc.io.DataStreamFromComStream
未能转换 com.ms.wfc.io.File.createDirectory
未能转换 com.ms.wfc.io.File.File
未能转换 com.ms.wfc.io.File.getDirectory
未能转换 com.ms.wfc.io.File.GetFiles
未能转换 com.ms.wfc.io.File.getName
未能转换 com.ms.wfc.io.File.handle
未能转换 com.ms.wfc.io.File.openStandardError
未能转换 com.ms.wfc.io.File.openStandardInput
未能转换 com.ms.wfc.io.File.openStandardOutput
未能转换 com.ms.wfc.io.File.readCore
未能转换 com.ms.wfc.io.File.writeCore

未能转换 com.ms.wfc.io.FileEnumerator.close
未能转换 com.ms.wfc.io.FileEnumerator.FileEnumerator
未能转换 com.ms.wfc.io.FileEnumerator.finalize
未能转换 com.ms.wfc.io.FileEnumerator.getAttributes
未能转换 com.ms.wfc.io.IDataStream.getComStream
未能转换 com.ms.wfc.io.FileEnumerator.getSize
未能转换 com.ms.wfc.io.IDataStream.readStringNull
未能转换 com.ms.wfc.io.IDataStream.readStringNullAnsi
未能转换 com.ms.wfc.io.IDataStream.readUTF
未能转换 com.ms.wfc.io.IDataStream.writeStringChars
未能转换 com.ms.wfc.io.IDataStream.writeStringCharsAnsi
未能转换 com.ms.wfc.io.IDataStream.writeStringNull
未能转换 com.ms.wfc.io.IDataStream.writeStringNullAnsi
未能转换 com.ms.wfc.io.IDataStream.writeUTF
未能转换 com.ms.wfc.io.IDataStreamProvider
未能转换 com.ms.wfc.io.MemoryStream.MemoryStream
未能转换 com.ms.wfc.io.MemoryStream.readCore
未能转换 com.ms.wfc.io.MemoryStream.readStringCharsAnsi
未能转换 com.ms.wfc.io.MemoryStream.readStringNull
未能转换 com.ms.wfc.io.MemoryStream.readStringNullAnsi
未能转换 com.ms.wfc.io.MemoryStream.writeCore
未能转换 com.ms.wfc.io.MemoryStream.writeTo
未能转换 com.ms.wfc.io.Reader.Reader
未能转换 com.ms.wfc.io.TextReader.TextReader(ByteStream)
未能转换 com.ms.wfc.io.TextReader.TextReader(ByteStream, int)
未能转换 com.ms.wfc.io.TextReader.TextReader(ByteStream, int, int)
未能转换 com.ms.wfc.io.WinIOException.getErrorCode
未能转换 com.ms.wfc.io.WinIOException.WinIOException
未能转换 com.ms.wfc.io.Writer.Writer

Com.ms.wfc.ole32 错误信息

未能转换 com.ms.wfc.ole32.DBBINDING.pTypeInfo
未能转换 com.ms.wfc.ole32.DBCOLUMNINFO.pTypeInfo
未能转换 com.ms.wfc.ole32.IEnumFORMATETC.iid
未能转换 com.ms.wfc.ole32.IEnumSTATDATA.iid
未能转换 com.ms.wfc.ole32.ILockBytes.iid
未能转换 com.ms.wfc.ole32.ICollection.iid
未能转换 com.ms.wfc.ole32.IOleDataFormat.iid
未能转换 com.ms.wfc.ole32.IOleDataObject.iid
未能转换 com.ms.wfc.ole32.IOleDataObjectWithIStream.iid
未能转换 com.ms.wfc.ole32.IOleDropSource.iid
未能转换 com.ms.wfc.ole32.IOleDropTarget.iid
未能转换 com.ms.wfc.ole32.IPersist.iid
未能转换 com.ms.wfc.ole32.IPersistStream.iid
未能转换 com.ms.wfc.ole32.IPicture.iid
未能转换 com.ms.wfc.ole32.IStorage.iid

Com.ms.wfc.ui 错误信息

未能转换 com.ms.wfc.ui.AnchorEditor.editValue
未能转换 com.ms.wfc.ui.AnchorEditor.getConstantName
未能转换 com.ms.wfc.ui.AnchorEditor.getTextFromValue
未能转换 com.ms.wfc.ui.Animation
未能转换 com.ms.wfc.ui.AnimationFileNameEditor
未能转换 com.ms.wfc.ui.AutoSizeEvent
未能转换 com.ms.wfc.ui.AutoSizeEventHandler
未能转换 com.ms.wfc.ui.AxHost.AboutBoxDelegate
未能转换 com.ms.wfc.ui.AxHost.AxPropertyInfo
未能转换 com.ms.wfc.ui.AxHost.begin
未能转换 com.ms.wfc.ui.AxHost.DispidAttribute
未能转换 com.ms.wfc.ui.AxHost.fireOnClick
未能转换 com.ms.wfc.ui.AxHost.fireOnDblClick
未能转换 com.ms.wfc.ui.AxHost.fireOnKeyDown
未能转换 com.ms.wfc.ui.AxHost.fireOnKeyPress
未能转换 com.ms.wfc.ui.AxHost.fireOnKeyUp
未能转换 com.ms.wfc.ui.AxHost.fireOnMouseDown
未能转换 com.ms.wfc.ui.AxHost.fireOnMouseMove
未能转换 com.ms.wfc.ui.AxHost.fireOnMouseUp
未能转换 com.ms.wfc.ui.AxHost.Flags
未能转换 com.ms.wfc.ui.AxHost.getAmbientProperty
未能转换 com.ms.wfc.ui.AxHost.getClientAttributes
未能转换 com.ms.wfc.ui.AxHost.getColorFromOleColor
未能转换 com.ms.wfc.ui.AxHost.getFontFromIFont
未能转换 com.ms.wfc.ui.AxHost.getFontFromIFontDisp
未能转换 com.ms.wfc.ui.AxHost.getIFontDispFromFont
未能转换 com.ms.wfc.ui.AxHost.getIFontFromFont
未能转换 com.ms.wfc.ui.AxHost.getIPictureDispFromPicture
未能转换 com.ms.wfc.ui.AxHost.getIPictureFromPicture
未能转换 com.ms.wfc.ui.AxHost.getOleColorFromColor
未能转换 com.ms.wfc.ui.AxHost.getPictureFromIPicture
未能转换 com.ms.wfc.ui.AxHost.getPictureFromIPictureDisp
未能转换 com.ms.wfc.ui.AxHost.propertyChanged
未能转换 com.ms.wfc.ui.AxHost.setAboutBoxDelegate
未能转换 com.ms.wfc.ui.AxHost.setLevel
未能转换 com.ms.wfc.ui.AxHost.shouldPersistContainingForm
未能转换 com.ms.wfc.ui.AxHost.State.save

未能转换 com.ms.wfc.ui.AxHost.State.State
未能转换 com.ms.wfc.ui.AxPersist
未能转换 com.ms.wfc.ui.Bitmap.Bitmap
未能转换 com.ms.wfc.ui.Bitmap.Bitmap(Bitmap, Bitmap)
未能转换 com.ms.wfc.ui.Bitmap.Bitmap(int)
未能转换 com.ms.wfc.ui.Bitmap.Bitmap(int, int, int, int, short[])
未能转换 com.ms.wfc.ui.Bitmap.Bitmap(IPicture)
未能转换 com.ms.wfc.ui.Bitmap.Bitmap(Palette)
未能转换 com.ms.wfc.ui.Bitmap.copyHandle
未能转换 com.ms.wfc.ui.Bitmap.destroyHandle
未能转换 com.ms.wfc.ui.Bitmap.drawStretchTo
未能转换 com.ms.wfc.ui.Bitmap.drawTo
未能转换 com.ms.wfc.ui.Bitmap.getColorMask
未能转换 com.ms.wfc.ui.Bitmap.getGraphics
未能转换 com.ms.wfc.ui.Bitmap.getHandle
未能转换 com.ms.wfc.ui.Bitmap.getMonochromeMask
未能转换 com.ms.wfc.ui.Bitmap.getPICTDESC
未能转换 com.ms.wfc.ui.Bitmap.getTransparent
未能转换 com.ms.wfc.ui.Bitmap.getTransparentColor
未能转换 com.ms.wfc.ui.Bitmap.initialize
未能转换 com.ms.wfc.ui.Brush.Brush(Bitmap)
未能转换 com.ms.wfc.ui.Brush.Brush(int)
未能转换 com.ms.wfc.ui.Brush.copyHandle
未能转换 com.ms.wfc.ui.Brush.destroyHandle
未能转换 com.ms.wfc.ui.Brush.getHandle
未能转换 com.ms.wfc.ui.Brush.HALFTONE
未能转换 com.ms.wfc.ui.BrushStyle.HOLLOW
未能转换 com.ms.wfc.ui.BrushStyle.PATTERN
未能转换 com.ms.wfc.ui.Button.propertyChanged
未能转换 com.ms.wfc.ui.CheckBox.getGroupValue
未能转换 com.ms.wfc.ui.CheckBox.setGroupValue
未能转换 com.ms.wfc.ui.CheckBox.setTextAlign
未能转换 com.ms.wfc.ui.CheckedListBox.propertyChanged
未能转换 com.ms.wfc.ui.Color.Color
未能转换 com.ms.wfc.ui.Color.Editor.ColorPicker
未能转换 com.ms.wfc.ui.Color.Editor.ColorPicker.Palette
未能转换 com.ms.wfc.ui.Color.Editor.ColorPicker.Palette.CustomColorDialog
未能转换 com.ms.wfc.ui.Color.fromCMYK
未能转换 com.ms.wfc.ui.Color.getConstructorArgs
未能转换 com.ms.wfc.ui.Color.toString

未能转换 com.ms.wfc.ui.ColumnClickEventHandler.ColumnClickEventHandler
未能转换 com.ms.wfc.ui.ComboBox.propertyChanged
未能转换 com.ms.wfc.ui.ContentAlignment.toTextFormat
未能转换 com.ms.wfc.ui.Control.getTopLevel
未能转换 com.ms.wfc.ui.Control.getVisible
未能转换 com.ms.wfc.ui.Control.invokeAsync
未能转换 com.ms.wfc.ui.Control.PROP_BACKCOLOR
未能转换 com.ms.wfc.ui.Control.PROP_CURSOR
未能转换 com.ms.wfc.ui.Control.PROP_ENABLED
未能转换 com.ms.wfc.ui.Control.PROP_FONT
未能转换 com.ms.wfc.ui.Control.PROP_FORECOLOR
未能转换 com.ms.wfc.ui.Control.PROP_LOCATION
未能转换 com.ms.wfc.ui.Control.PROP_PARENTBACKCOLOR
未能转换 com.ms.wfc.ui.Control.PROP_PARENTFONT
未能转换 com.ms.wfc.ui.Control.PROP_PARENTFORECOLOR
未能转换 com.ms.wfc.ui.Control.PROP_SIZE
未能转换 com.ms.wfc.ui.Control.PROP_TEXT
未能转换 com.ms.wfc.ui.Control.PROP_VISIBLE
未能转换 com.ms.wfc.ui.Control.propertyChanged
未能转换 com.ms.wfc.ui.Control.sendMessage
未能转换 com.ms.wfc.ui.Control.setTabIndex
未能转换 com.ms.wfc.ui.Control.setTopLevel
未能转换 com.ms.wfc.ui.Control.STYLE_ACCEPTSCHILDREN
未能转换 com.ms.wfc.ui.CoordinateSystem
未能转换 com.ms.wfc.ui.Cursor.Cursor
未能转换 com.ms.wfc.ui.Cursor.destroyHandle
未能转换 com.ms.wfc.ui.Cursor.getPICTDESC
未能转换 com.ms.wfc.ui.Cursor.initialize
未能转换 com.ms.wfc.ui.DateBoldEventHandler.DateBoldEventHandler
未能转换 com.ms.wfc.ui.DateRangeEventHandler.DateRangeEventHandler
未能转换 com.ms.wfc.ui.DateTimeFormatEvent
未能转换 com.ms.wfc.ui.DateTimeFormatEventHandler
未能转换 com.ms.wfc.ui.DateTimeFormatQueryEvent
未能转换 com.ms.wfc.ui.DateTimeFormatQueryEventHandler
未能转换 com.ms.wfc.ui.DateTimePicker.addOnFormat
未能转换 com.ms.wfc.ui.DateTimePicker.addOnFormatQuery
未能转换 com.ms.wfc.ui.DateTimePicker.addOnUserString
未能转换 com.ms.wfc.ui.DateTimePicker.getAllowUserString
未能转换 com.ms.wfc.ui.DateTimePicker.getMaxDate
未能转换 com.ms.wfc.ui.DateTimePicker.getMinDate

未能转换 com.ms.wfc.ui.DateTimePicker.onFormat
未能转换 com.ms.wfc.ui.DateTimePicker.onFormatQuery
未能转换 com.ms.wfc.ui.DateTimePicker.onUserString
未能转换 com.ms.wfc.ui.DateTimePicker.removeOnFormat
未能转换 com.ms.wfc.ui.DateTimePicker.removeOnFormatQuery
未能转换 com.ms.wfc.ui.DateTimePicker.removeOnUserString
未能转换 com.ms.wfc.ui.DateTimePicker.setAllowUserString
未能转换 com.ms.wfc.ui.DateTimeUserStringEvent
未能转换 com.ms.wfc.ui.DateTimeUserStringEventHandler
未能转换 com.ms.wfc.ui.DateTimeWmKeyDownEvent.DateTimeWmKeyDownEvent
未能转换 com.ms.wfc.ui.DateTimeWmKeyDownEvent.format
未能转换 com.ms.wfc.ui.DateTimeWmKeyDownEvent.time
未能转换 com.ms.wfc.ui.DateTimeWmKeyDownEventHandler.DateTimeWmKeyDownEventHandler
未能转换 com.ms.wfc.ui.Dimensions.Editor
未能转换 com.ms.wfc.ui.Dimensions.getConstructorArgs
未能转换 com.ms.wfc.ui.Dimensions.save
未能转换 com.ms.wfc.ui.DockEditor.editValue
未能转换 com.ms.wfc.ui.DockEditor.getConstantName
未能转换 com.ms.wfc.ui.DockEditor.getTextFromValue
未能转换 com.ms.wfc.ui.DocumentReadyEvent
未能转换 com.ms.wfc.ui.DocumentReadyEventHandler
未能转换 com.ms.wfc.ui.DocumentReadyEventHandler.DocumentReadyEventHandler
未能转换 com.ms.wfc.ui.DragEventHandler.DragEventHandler
未能转换 com.ms.wfc.ui.DrawItemEventHandler.DrawItemEventHandler
未能转换 com.ms.wfc.ui.Edit.propertyChanged
未能转换 com.ms.wfc.ui.Edit.setTextAlign
未能转换 com.ms.wfc.ui.Edit.undo
未能转换 com.ms.wfc.ui.Editor.createExtensionsString
未能转换 com.ms.wfc.ui.Editor.createFilterEntry
未能转换 com.ms.wfc.ui.Editor.getExtensions
未能转换 com.ms.wfc.ui.Editor.getFileDialogDescription
未能转换 com.ms.wfc.ui.Editor.loadFromStream
未能转换 com.ms.wfc.ui.EraseBackgroundEvent
未能转换 com.ms.wfc.ui.FileDialog.promptFileCreate
未能转换 com.ms.wfc.ui.FileDialog.promptFileNotFound
未能转换 com.ms.wfc.ui.FileDialog.promptFileOverwrite
未能转换 com.ms.wfc.ui.FloodFillType
未能转换 com.ms.wfc.ui.Font.ANSI_VAR
未能转换 com.ms.wfc.ui.Font.destroyHandle
未能转换 com.ms.wfc.ui.Font.DEVICE_DEFAULT

未能转换 com.ms.wfc.ui.Font.equalsBase
未能转换 com.ms.wfc.ui.Font.Font
未能转换 com.ms.wfc.ui.Font.getCharacterSet
未能转换 com.ms.wfc.ui.Font.getConstructorArgs
未能转换 com.ms.wfc.ui.Font.getFontFamily
未能转换 com.ms.wfc.ui.Font.getFontMetrics
未能转换 com.ms.wfc.ui.Font.getOrientation
未能转换 com.ms.wfc.ui.Font.getPitch
未能转换 com.ms.wfc.ui.Font.getStock
未能转换 com.ms.wfc.ui.Font.OEM_FIXED
未能转换 com.ms.wfc.ui.Font.save
未能转换 com.ms.wfc.ui.Font.SYSTEM
未能转换 com.ms.wfc.ui.Font.SYSTEM_FIXED
未能转换 com.ms.wfc.ui.FontDescriptor
未能转换 com.ms.wfc.ui.FontDevice
未能转换 com.ms.wfc.ui.FontDialog.getFontDevice
未能转换 com.ms.wfc.ui.FontDialog.getPrinterDC
未能转换 com.ms.wfc.ui.FontDialog.getScalableOnly
未能转换 com.ms.wfc.ui.FontDialog.getTrueTypeOnly
未能转换 com.ms.wfc.ui.FontDialog.getWysiwyg
未能转换 com.ms.wfc.ui.FontDialog.hookProc
未能转换 com.ms.wfc.ui.FontDialog.setFontDevice
未能转换 com.ms.wfc.ui.FontDialog.setPrinterDC
未能转换 com.ms.wfc.ui.FontDialog.setScalableOnly
未能转换 com.ms.wfc.ui.FontDialog.setTrueTypeOnly
未能转换 com.ms.wfc.ui.FontDialog.setWysiwyg
未能转换 com.ms.wfc.ui.FontFamily
未能转换 com.ms.wfc.ui.FontMetrics
未能转换 com.ms.wfc.ui.FontMetrics.ascent
未能转换 com.ms.wfc.ui.FontMetrics.charSet
未能转换 com.ms.wfc.ui.FontMetrics.descent
未能转换 com.ms.wfc.ui.FontMetrics.FontMetrics
未能转换 com.ms.wfc.ui.FontMetrics.height
未能转换 com.ms.wfc.ui.FontPitch
未能转换 com.ms.wfc.ui.FontSize.CELLHEIGHT
未能转换 com.ms.wfc.ui.FontSize.CENTIMETERS
未能转换 com.ms.wfc.ui.FontSize.CHARACTERHEIGHT
未能转换 com.ms.wfc.ui.FontSize.EM
未能转换 com.ms.wfc.ui.FontSize.EX
未能转换 com.ms.wfc.ui.FontType

未能转换 com.ms.wfc.ui.FontWeight.EXTRALIGHT
未能转换 com.ms.wfc.ui.FontWeight.LIGHT
未能转换 com.ms.wfc.ui.FontWeight.THIN
未能转换 com.ms.wfc.ui.Form.checkCloseDialog
未能转换 com.ms.wfc.ui.Form.FORMSTATE_AUTOSCALING
未能转换 com.ms.wfc.ui.Form.FORMSTATE_AUTOSCROLLING
未能转换 com.ms.wfc.ui.Form.FORMSTATE_BORDERSTYLE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_CONTROLBOX
未能转换 com.ms.wfc.ui.Form.FORMSTATE_HELPBUTTON
未能转换 com.ms.wfc.ui.Form.FORMSTATE_HSCROLLVISIBLE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_KEYPREVIEW
未能转换 com.ms.wfc.ui.Form.FORMSTATE_MAXIMIZEBOX
未能转换 com.ms.wfc.ui.Form.FORMSTATE_MINIMIZEBOX
未能转换 com.ms.wfc.ui.Form.FORMSTATE_PALETTE MODE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_SETCLIENTSIZE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_SHOWWINDOWONCREATE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_STARTPOS
未能转换 com.ms.wfc.ui.Form.FORMSTATE_TASKBAR
未能转换 com.ms.wfc.ui.Form.FORMSTATE_TOPMOST
未能转换 com.ms.wfc.ui.Form.FORMSTATE_USERHASSCROLLED
未能转换 com.ms.wfc.ui.Form.FORMSTATE_VSCROLLVISIBLE
未能转换 com.ms.wfc.ui.Form.FORMSTATE_WINDOWSTATE
未能转换 com.ms.wfc.ui.Form.getFormState
未能转换 com.ms.wfc.ui.Form.getPalette
未能转换 com.ms.wfc.ui.Form.getPaletteMode
未能转换 com.ms.wfc.ui.Form.getPaletteSource
未能转换 com.ms.wfc.ui.Form.hasFormState
未能转换 com.ms.wfc.ui.Form.notifyPaletteChange
未能转换 com.ms.wfc.ui.Form.onMDIChildActivate
未能转换 com.ms.wfc.ui.Form.onNewPalette
未能转换 com.ms.wfc.ui.Form.propertyChanged
未能转换 com.ms.wfc.ui.Form.setScaleBaseSize
未能转换 com.ms.wfc.ui.Form.setBorderStyle
未能转换 com.ms.wfc.ui.Form.setControlBox
未能转换 com.ms.wfc.ui.Form.setFormState
未能转换 com.ms.wfc.ui.Form.setMaximizeBox
未能转换 com.ms.wfc.ui.Form.setMinimizeBox
未能转换 com.ms.wfc.ui.Form.setNewControls
未能转换 com.ms.wfc.ui.Form.setPaletteMode
未能转换 com.ms.wfc.ui.Form.setPaletteSource

未能转换 com.ms.wfc.ui.Form.setShowInTaskbar
未能转换 com.ms.wfc.ui.Form.setStartPosition
未能转换 com.ms.wfc.ui.FormPaletteMode
未能转换 com.ms.wfc.ui.GiveFeedbackEventHandler.GiveFeedbackEventHandler
未能转换 com.ms.wfc.ui.Graphics.drawArc
未能转换 com.ms.wfc.ui.Graphics.drawChord
未能转换 com.ms.wfc.ui.Graphics.drawPie
未能转换 com.ms.wfc.ui.Graphics.drawString
未能转换 com.ms.wfc.ui.Graphics.floodFill
未能转换 com.ms.wfc.ui.Graphics.getCoordinateSystem
未能转换 com.ms.wfc.ui.Graphics.getDeviceOrigin
未能转换 com.ms.wfc.ui.Graphics.getDevicePoint
未能转换 com.ms.wfc.ui.Graphics.getDeviceScale
未能转换 com.ms.wfc.ui.Graphics.getFontDescriptors
未能转换 com.ms.wfc.ui.Graphics.getHandle
未能转换 com.ms.wfc.ui.Graphics.getLogicalPoint
未能转换 com.ms.wfc.ui.Graphics.getLogicalSizeX
未能转换 com.ms.wfc.ui.Graphics.getLogicalSizeY
未能转换 com.ms.wfc.ui.Graphics.getOpaque
未能转换 com.ms.wfc.ui.Graphics.getPageOrigin
未能转换 com.ms.wfc.ui.Graphics.getPageScale
未能转换 com.ms.wfc.ui.Graphics.getPhysicalSizeX
未能转换 com.ms.wfc.ui.Graphics.getPhysicalSizeY
未能转换 com.ms.wfc.ui.Graphics.getPixel
未能转换 com.ms.wfc.ui.Graphics.getTextSize
未能转换 com.ms.wfc.ui.Graphics.getTextSpace
未能转换 com.ms.wfc.ui.Graphics.Graphics
未能转换 com.ms.wfc.ui.Graphics.invert
未能转换 com.ms.wfc.ui.Graphics.renderPalette
未能转换 com.ms.wfc.ui.Graphics.scroll
未能转换 com.ms.wfc.ui.Graphics.setCoordinateOrigin
未能转换 com.ms.wfc.ui.Graphics.setCoordinateScale
未能转换 com.ms.wfc.ui.Graphics.setCoordinateSystem
未能转换 com.ms.wfc.ui.Graphics.setHandle
未能转换 com.ms.wfc.ui.Graphics.setOpaque
未能转换 com.ms.wfc.ui.Graphics.setPixel
未能转换 com.ms.wfc.ui.Graphics.setTextSpace
未能转换 com.ms.wfc.ui.Help.Help
未能转换 com.ms.wfc.ui.HelpEvent.component
未能转换 com.ms.wfc.ui.HelpEvent.contextId

未能转换 com.ms.wfc.ui.HelpEvent.contextType
未能转换 com.ms.wfc.ui.HelpEvent.controlId
未能转换 com.ms.wfc.ui.HelpEvent.HelpEvent(int, int, IComonent, int, Point)
未能转换 com.ms.wfc.ui.HelpEvent.HelpEvent(Object, int, int, IComonent, Int, Point)
未能转换 com.ms.wfc.ui.HelpEventHandler.HelpEventHandler
未能转换 com.ms.wfc.ui.HelpFileFileNameEditor
未能转换 com.ms.wfc.ui.HelpProvider.shouldPersistShowHelp
未能转换 com.ms.wfc.ui.HTMLControl
未能转换 com.ms.wfc.ui.HTMLControl.add
未能转换 com.ms.wfc.ui.HTMLControl.addOnDocumentReady
未能转换 com.ms.wfc.ui.HTMLControl.getAmbientProperty
未能转换 com.ms.wfc.ui.HTMLControl.HTMLControl
未能转换 com.ms.wfc.ui.HTMLControl.onDocumentReady
未能转换 com.ms.wfc.ui.HTMLControl.propertyChanged
未能转换 com.ms.wfc.ui.HTMLControl.removeOnDocumentReady
未能转换 com.ms.wfc.ui.HTMLControl.setBoundElements
未能转换 com.ms.wfc.ui.HTMLControl.setNewHTMLElements
未能转换 com.ms.wfc.ui.HTMLControl.setURL
未能转换 com.ms.wfc.ui.IActiveXCustomPropertyDialog
未能转换 com.ms.wfc.ui.Icon.getPICTDESC
未能转换 com.ms.wfc.ui.Icon.Icon
未能转换 com.ms.wfc.ui.Icon.initialize
未能转换 com.ms.wfc.ui.Icon.loadPicture
未能转换 com.ms.wfc.ui.IHandleHook
未能转换 com.ms.wfc.ui.Image.copyHandle
未能转换 com.ms.wfc.ui.Image.dirty
未能转换 com.ms.wfc.ui.Image.drawStretchTo
未能转换 com.ms.wfc.ui.Image.drawTo
未能转换 com.ms.wfc.ui.ImagegetExtension
未能转换 com.ms.wfc.ui.Image.getHandle
未能转换 com.ms.wfc.ui.Image.getPICTDESC
未能转换 com.ms.wfc.ui.Image.getPicture
未能转换 com.ms.wfc.ui.Image.Image
未能转换 com.ms.wfc.ui.Image.initialize
未能转换 com.ms.wfc.ui.Image.loadImage
未能转换 com.ms.wfc.ui.Image.loadPicture
未能转换 com.ms.wfc.ui.Image.PictureList
未能转换 com.ms.wfc.ui.ImageIndexEditor
未能转换 com.ms.wfc.ui.ImageList.createHandle
未能转换 com.ms.wfc.ui.ImageList.destroyHandle

未能转换 com.ms.wfc.ui.ImageList.getBackColor
未能转换 com.ms.wfc.ui.ImageList.getIcon
未能转换 com.ms.wfc.ui.ImageList.getMaskColor
未能转换 com.ms.wfc.ui.ImageList.getUseMask
未能转换 com.ms.wfc.ui.ImageList.ImageList
未能转换 com.ms.wfc.ui.ImageList.recreateHandle
未能转换 com.ms.wfc.ui.ImageList.SetBackColor
未能转换 com.ms.wfc.ui.ImageList.setImages
未能转换 com.ms.wfc.ui.ImageList.setMaskColor
未能转换 com.ms.wfc.ui.ImageList.setUseMask
未能转换 com.ms.wfc.ui.ImageListStreamergetExtension
未能转换 com.ms.wfc.ui.ImageListStreamer.ImageListStreamer
未能转换 com.ms.wfc.ui.ImageListStreamer.save
未能转换 com.ms.wfc.ui.InputLangChangeEventHandler.InputLangChangeEventHandler
未能转换 com.ms.wfc.ui.InputLangChangeEvent.InputLangChangeEvent
未能转换 com.ms.wfc.ui.InputLangChangeRequestEventHandler.InputLangChangeRequestEventHandler
未能转换 com.ms.wfc.ui.ISelectionService.setSelectionStyle
未能转换 com.ms.wfc.ui.ItemCheckEventHandler.ItemCheckEventHandler
未能转换 com.ms.wfc.ui.ItemDragEventHandler.ItemDragEventHandler
未能转换 com.ms.wfc.ui.KeyEvent(KeyEvent
未能转换 com.ms.wfc.ui.KeyEventHandler.KeyEventHandler
未能转换 com.ms.wfc.ui.KeyPressEvent.KeyPressEvent
未能转换 com.ms.wfc.ui.KeyPressEventHandler.KeyPressEventHandler
未能转换 com.ms.wfc.ui.Label.propertyChanged
未能转换 com.ms.wfc.ui.Label.setTextAlign
未能转换 com.ms.wfc.ui.LabelEditEventHandler.LabelEditEventHandler
未能转换 com.ms.wfc.ui.LayoutEventHandler.LayoutEventHandler
未能转换 com.ms.wfc.ui.ListItem.getConstructorArgs
未能转换 com.ms.wfc.ui.ListItem.ListItem
未能转换 com.ms.wfc.ui.ListItem.ListItem(Stream)
未能转换 com.ms.wfc.ui.ListItem.ListItem(String, int, String[]])
未能转换 com.ms.wfc.ui.ListItem.ListItem(String, String[]])
未能转换 com.ms.wfc.ui.ListItem.save
未能转换 com.ms.wfc.ui.ListItem.setSubItem
未能转换 com.ms.wfc.ui.MDIWindowDialog
未能转换 com.ms.wfc.ui.MeasureItemEventHandler.MeasureItemEventHandler
未能转换 com.ms.wfc.ui.Menu.Menu
未能转换 com.ms.wfc.ui.Menu.mergeMenu
未能转换 com.ms.wfc.ui.MenuItem.setChecked
未能转换 com.ms.wfc.ui.Metafile.copyHandle

未能转换 com.ms.wfc.ui.Metafile.destroyHandle
未能转换 com.ms.wfc.ui.Metafile.drawStretchTo
未能转换 com.ms.wfc.ui.Metafile.drawTo
未能转换 com.ms.wfc.ui.Metafile.getHandle
未能转换 com.ms.wfc.ui.Metafile.getPICTDESC
未能转换 com.ms.wfc.ui.Metafile.getRenderedSize
未能转换 com.ms.wfc.ui.Metafile.initialize
未能转换 com.ms.wfc.ui.Metafile.Metafile
未能转换 com.ms.wfc.ui.MonthCalendar.propertyChanged
未能转换 com.ms.wfc.ui.MouseEvent.MouseEvent
未能转换 com.ms.wfc.ui.MouseEventHandler.MouseEventHandler
未能转换 com.ms.wfc.ui.NodeLabelEditEventHandler.NodeLabelEditEventHandler
未能转换 com.ms.wfc.ui.PaintEvent.graphics
未能转换 com.ms.wfc.ui.PaintEventHandler.PaintEventHandler
未能转换 com.ms.wfc.ui.Palette
未能转换 com.ms.wfc.ui.Palette.clone
未能转换 com.ms.wfc.ui.Palette.copyHandle
未能转换 com.ms.wfc.ui.Palette.destroyHandle
未能转换 com.ms.wfc.ui.Palette.dispose
未能转换 com.ms.wfc.ui.Palette.equals
未能转换 com.ms.wfc.ui.Palette.finalize
未能转换 com.ms.wfc.ui.Palette.getHalftonePalette
未能转换 com.ms.wfc.ui.Palette.getHandle
未能转换 com.ms.wfc.ui.Palette.getPaletteSupported
未能转换 com.ms.wfc.ui.Palette.Palette
未能转换 com.ms.wfc.ui.Pen.copyHandle
未能转换 com.ms.wfc.ui.Pen.destroyHandle
未能转换 com.ms.wfc.ui.Pen.getHandle
未能转换 com.ms.wfc.ui.Pen.Pen
未能转换 com.ms.wfc.ui.PenEntry
未能转换 com.ms.wfc.ui.PenStyle.INSIDEFRAME
未能转换 com.ms.wfc.ui.PenStyle.NULL
未能转换 com.ms.wfc.ui.PictureBox.setImage
未能转换 com.ms.wfc.ui.Point.Editor
未能转换 com.ms.wfc.ui.Point.getConstructorArgs
未能转换 com.ms.wfc.ui.Point.Point
未能转换 com.ms.wfc.ui.Point.save
未能转换 com.ms.wfc.ui.PopulatedMenusEditor
未能转换 com.ms.wfc.ui.ProgressBar.addValueChanged
未能转换 com.ms.wfc.ui.ProgressBar.removeOnValueChanged

未能转换 com.ms.wfc.ui.QueryContinueDragEventHandler.QueryContinueDragEventHandler
未能转换 com.ms.wfc.ui.RadioButton.setTextAlign
未能转换 com.ms.wfc.ui.Radix
未能转换 com.ms.wfc.ui.RasterOp
未能转换 com.ms.wfc.ui.ReadyStateEvent
未能转换 com.ms.wfc.ui.ReadyStateEventHandler
未能转换 com.ms.wfc.ui.Rebar.addBand
未能转换 com.ms.wfc.ui.Rebar.addOnAutoSize
未能转换 com.ms.wfc.ui.Rebar.addOnHeightChange
未能转换 com.ms.wfc.ui.Rebar.addOnLayoutChange
未能转换 com.ms.wfc.ui.Rebar.applyAutoSize
未能转换 com.ms.wfc.ui.Rebar.getBandBorders
未能转换 com.ms.wfc.ui.Rebar.getBands
未能转换 com.ms.wfc.ui.Rebar.getDoubleClickToggle
未能转换 com.ms.wfc.ui.Rebar.getFixedOrder
未能转换 com.ms.wfc.ui.Rebar.getOrientation
未能转换 com.ms.wfc.ui.Rebar.onAutoSize
未能转换 com.ms.wfc.ui.Rebar.onHeightChange
未能转换 com.ms.wfc.ui.Rebar.onLayoutChange
未能转换 com.ms.wfc.ui.Rebar.removeAllBands
未能转换 com.ms.wfc.ui.Rebar.removeBand
未能转换 com.ms.wfc.ui.Rebar.removeOnAutoSize
未能转换 com.ms.wfc.ui.Rebar.removeOnHeightChange
未能转换 com.ms.wfc.ui.Rebar.removeOnLayoutChange
未能转换 com.ms.wfc.ui.Rebar.setBandBorders
未能转换 com.ms.wfc.ui.Rebar.setBands
未能转换 com.ms.wfc.ui.Rebar.setComponentSite
未能转换 com.ms.wfc.ui.Rebar.setDoubleClickToggle
未能转换 com.ms.wfc.ui.Rebar.setFixedOrder
未能转换 com.ms.wfc.ui.Rebar.setNewControls
未能转换 com.ms.wfc.ui.Rebar.setOrientation
未能转换 com.ms.wfc.ui.RebarBand.getAllowVariableHeight
未能转换 com.ms.wfc.ui.RebarBand.getAllowVertical
未能转换 com.ms.wfc.ui.RebarBand.getBandBreak
未能转换 com.ms.wfc.ui.RebarBand getChildControl
未能转换 com.ms.wfc.ui.RebarBand.getChildEdge
未能转换 com.ms.wfc.ui.RebarBand.getFixedBitmap
未能转换 com.ms.wfc.ui.RebarBand.getGrowBy
未能转换 com.ms.wfc.ui.RebarBand.getHeaderWidth
未能转换 com.ms.wfc.ui.RebarBand.getIdealWidth

未能转换 com.ms.wfc.ui.RebarBand.getImageIndex
未能转换 com.ms.wfc.ui.RebarBand.getIndex
未能转换 com.ms.wfc.ui.RebarBand.getMaxInitialHeight
未能转换 com.ms.wfc.ui.RebarBand.getMinChildHeight
未能转换 com.ms.wfc.ui.RebarBand.getMinChildWidth
未能转换 com.ms.wfc.ui.RebarBand.getVisibleGripper
未能转换 com.ms.wfc.ui.RebarBand.maximize
未能转换 com.ms.wfc.ui.RebarBand.minimize
未能转换 com.ms.wfc.ui.RebarBand.setAllowVariableHeight
未能转换 com.ms.wfc.ui.RebarBand.setAllowVertical
未能转换 com.ms.wfc.ui.RebarBand.setBandBreak
未能转换 com.ms.wfc.ui.RebarBand.setChildControl
未能转换 com.ms.wfc.ui.RebarBand.setChildEdge
未能转换 com.ms.wfc.ui.RebarBand.setFixedBitmap
未能转换 com.ms.wfc.ui.RebarBand.setGrowBy
未能转换 com.ms.wfc.ui.RebarBand.setHeaderWidth
未能转换 com.ms.wfc.ui.RebarBand.setIdealWidth
未能转换 com.ms.wfc.ui.RebarBand.setImageIndex
未能转换 com.ms.wfc.ui.RebarBand.setIndex
未能转换 com.ms.wfc.ui.RebarBand.setMaxInitialHeight
未能转换 com.ms.wfc.ui.RebarBand.setMinChildHeight
未能转换 com.ms.wfc.ui.RebarBand.setMinChildWidth
未能转换 com.ms.wfc.ui.RebarBand.setVisibleGripper
未能转换 com.ms.wfc.ui.RebarBand.updateStyle
未能转换 com.ms.wfc.ui.Rectangle.Editor.Editor
未能转换 com.ms.wfc.ui.Rectangle.getConstructorArgs
未能转换 com.ms.wfc.ui.Rectangle.Rectangle
未能转换 com.ms.wfc.ui.Rectangle.save
未能转换 com.ms.wfc.ui.Rectangle.setBounds
未能转换 com.ms.wfc.ui.Rectangle.toRECT
未能转换 com.ms.wfc.ui.Region.copyHandle
未能转换 com.ms.wfc.ui.Region.createPolygonal
未能转换 com.ms.wfc.ui.Region.destroyHandle
未能转换 com.ms.wfc.ui.Region.getHandle
未能转换 com.ms.wfc.ui.RequestResizeEventHandler.RequestResizeEventHandler
未能转换 com.ms.wfc.ui.RichEdit.DLL_RICHEDIT
未能转换 com.ms.wfc.ui.RichEdit.getDelimiter
未能转换 com.ms.wfc.ui.RichEdit.getFollowPunctuation
未能转换 com.ms.wfc.ui.RichEdit.getIMEColor
未能转换 com.ms.wfc.ui.RichEdit.getIMEOptions

未能转换 com.ms.wfc.ui.RichEdit.getLeadPunctuation
未能转换 com.ms.wfc.ui.RichEdit.getOnHScroll
未能转换 com.ms.wfc.ui.RichEdit.getOnIMEChange
未能转换 com.ms.wfc.ui.RichEdit.getOnProtected
未能转换 com.ms.wfc.ui.RichEdit.getOnRequestResize
未能转换 com.ms.wfc.ui.RichEdit.getOnSelChange
未能转换 com.ms.wfc.ui.RichEdit.getOnVScroll
未能转换 com.ms.wfc.ui.RichEdit.getWordBreak
未能转换 com.ms.wfc.ui.RichEdit.getWordPunctuation
未能转换 com.ms.wfc.ui.RichEdit.moveToInsertionPoint
未能转换 com.ms.wfc.ui.RichEdit.RICHEDIT_CLASS10A
未能转换 com.ms.wfc.ui.RichEdit.RICHEDIT_CLASSA
未能转换 com.ms.wfc.ui.RichEdit.RICHEDIT_CLASSW
未能转换 com.ms.wfc.ui.RichEdit.RICHEDIT_DLL10
未能转换 com.ms.wfc.ui.RichEdit.RICHEDIT_DLL20
未能转换 com.ms.wfc.ui.RichEdit.setFollowPunctuation
未能转换 com.ms.wfc.ui.RichEdit.setIMEColor
未能转换 com.ms.wfc.ui.RichEdit.setIMEOptions
未能转换 com.ms.wfc.ui.RichEdit.setLeadPunctuation
未能转换 com.ms.wfc.ui.RichEdit.setWordBreak
未能转换 com.ms.wfc.ui.RichEdit.setWordPunctuation
未能转换 com.ms.wfc.ui.RichEdit.span
未能转换 com.ms.wfc.ui.RichEdit.WC_RICHEDIT
未能转换 com.ms.wfc.ui.RichEditIMEColor
未能转换 com.ms.wfc.ui.RichEditIMEOptions
未能转换 com.ms.wfc.ui.SameParentReferenceEditor
未能转换 com.ms.wfc.ui.SaveFileDialog.runFileDialog
未能转换 com.ms.wfc.ui.SaveFileDialog.setCreatePrompt
未能转换 com.ms.wfc.ui.Screen.MonitorEnumProc
未能转换 com.ms.wfc.ui.Screen.MONITORINFO
未能转换 com.ms.wfc.ui.Screen.MONITORINFOEX
未能转换 com.ms.wfc.ui.ScrollBar.propertyChanged
未能转换 com.ms.wfc.ui.ScrollEvent.ScrollEvent
未能转换 com.ms.wfc.ui.ScrollEventHandler.ScrollEventHandler
未能转换 com.ms.wfc.ui.SelectionChangedEventHandler.SelectionChangedEventHandler
未能转换 com.ms.wfc.ui.SelectionRange.Editor
未能转换 com.ms.wfc.ui.SelectionRange.getConstructorArgs
未能转换 com.ms.wfc.ui.SelectionStyle
未能转换 com.ms.wfc.ui.Shortcut.Editor
未能转换 com.ms.wfc.ui.Splitter.postFilterMessage

未能转换 com.ms.wfc.ui.SplitterEventHandler.SplitterEventHandler
未能转换 com.ms.wfc.ui.State.save
未能转换 com.ms.wfc.ui.State.State
未能转换 com.ms.wfc.ui.StatusBarDrawItemEventHandler.StatusBarDrawItemEventHandler
未能转换 com.ms.wfc.ui.StatusBarPanelClickEventHandler.StatusBarPanelClickEventHandler
未能转换 com.ms.wfc.ui.TabBase.getTCITEM
未能转换 com.ms.wfc.ui.TabBase.propertyChanged
未能转换 com.ms.wfc.ui.TabBase.setSelectedIndex
未能转换 com.ms.wfc.ui.TabStrip.insertTab
未能转换 com.ms.wfc.ui.TextFormat.BOTTOM
未能转换 com.ms.wfc.ui.TextFormat.EDITCONTROL
未能转换 com.ms.wfc.ui.TextFormat.ENDELLIPSIS
未能转换 com.ms.wfc.ui.TextFormat.EXPANDTABS
未能转换 com.ms.wfc.ui.TextFormat.HORIZONTALCENTER
未能转换 com.ms.wfc.ui.TextFormat.LEFT
未能转换 com.ms.wfc.ui.TextFormat.NOPREFIX
未能转换 com.ms.wfc.ui.TextFormat.PATHELLIPSIS
未能转换 com.ms.wfc.ui.TextFormat.RIGHT
未能转换 com.ms.wfc.ui.TextFormat.RIGHTTOLEFT
未能转换 com.ms.wfc.ui.TextFormat.SINGLELINE
未能转换 com.ms.wfc.ui.TextFormat.TextFormat
未能转换 com.ms.wfc.ui.TextFormat.TOP
未能转换 com.ms.wfc.ui.TextFormat.valid
未能转换 com.ms.wfc.ui.TextFormat.VERTICALCENTER
未能转换 com.ms.wfc.ui.TextFormat.WORDBREAK
未能转换 com.ms.wfc.ui.Toolbar.propertyChanged
未能转换 com.ms.wfc.ui.ToolBar.TextAlign.RIGHT
未能转换 com.ms.wfc.ui.ToolBar.TextAlign.UNDERNEATH
未能转换 com.ms.wfc.ui.ToolTip.shouldPersistAutomaticDelay
未能转换 com.ms.wfc.ui.ToolTip.shouldPersistAutoPopDelay
未能转换 com.ms.wfc.ui.ToolTip.shouldPersistInitialDelay
未能转换 com.ms.wfc.ui.ToolTip.shouldPersistReshowDelay
未能转换 com.ms.wfc.ui.TrackBar.propertyChanged
未能转换 com.ms.wfc.uiTreeNode.getConstructorArgs
未能转换 com.ms.wfc.uiTreeNode.save
未能转换 com.ms.wfc.uiTreeNodeTreeNode
未能转换 com.ms.wfc.uiTreeNodeEditDialog
未能转换 com.ms.wfc.uiTreeNodeEditor
未能转换 com.ms.wfc.ui.TreeView.shouldPersistIndent
未能转换 com.ms.wfc.ui.TreeViewCancelEventHandler.TreeViewCancelEventHandler

未能转换 com.ms.wfc.ui.TreeViewEventHandler.TreeViewEventHandler
未能转换 com.ms.wfc.uiUpDown.getAcceleration
未能转换 com.ms.wfc.uiUpDown.getAutoBuddy
未能转换 com.ms.wfc.uiUpDown.getAutoSize
未能转换 com.ms.wfc.uiUpDown.getBuddyControl
未能转换 com.ms.wfc.uiUpDown.getHorizontal
未能转换 com.ms.wfc.uiUpDown.getModifyBuddy
未能转换 com.ms.wfc.uiUpDown.getRadix
未能转换 com.ms.wfc.uiUpDown.getWrap
未能转换 com.ms.wfc.uiUpDown.onCreateHandle
未能转换 com.ms.wfc.uiUpDown.setAcceleration
未能转换 com.ms.wfc.uiUpDown.setAutoBuddy
未能转换 com.ms.wfc.uiUpDown.setAutoSize
未能转换 com.ms.wfc.uiUpDown.setBuddyControl
未能转换 com.ms.wfc.uiUpDown.setHorizontal
未能转换 com.ms.wfc.uiUpDown.setModifyBuddy
未能转换 com.ms.wfc.uiUpDown.setRadix
未能转换 com.ms.wfc.uiUpDown.setWrap
未能转换 com.ms.wfc.uiUpDown.shouldPersistBuddyControl
未能转换 com.ms.wfc.uiUpDownAcceleration
未能转换 com.ms.wfc.uiUpDownAcceleration.Editor
未能转换 com.ms.wfc.uiUpDownAlignment.MANUAL
未能转换 com.ms.wfc.uiUpDownChangedEvent.delta
未能转换 com.ms.wfc.uiUpDownChangedEvent.value
未能转换 com.ms.wfc.uiUpDownChangedEvent.Handler.UpDownChangedEventHandler

Com.ms.wfc.util 错误信息

未能转换 com.ms.wfc.util.ArrayEnumerator.hasMoreItems
未能转换 com.ms.wfc.util.Debug.addOnDisplayAssert
未能转换 com.ms.wfc.util.Debug.addOnDisplayMessage
未能转换 com.ms.wfc.util.Debug.Debug
未能转换 com.ms.wfc.util.Debug.displaySwitches
未能转换 com.ms.wfc.util.Debug.getSwitchListText
未能转换 com.ms.wfc.util.Debug.printEnumIf
未能转换 com.ms.wfc.util.Debug.printExceptionIf
未能转换 com.ms.wfc.util.Debug.printf
未能转换 com.ms.wfc.util.Debug.printlnIf
未能转换 com.ms.wfc.util.Debug.printObjectIf
未能转换 com.ms.wfc.util.Debug.printSwitchList
未能转换 com.ms.wfc.util.Debug.printStackTraceIf
未能转换 com.ms.wfc.util.Debug.removeOnDisplayAssert
未能转换 com.ms.wfc.util.Debug.removeOnDisplayMessage
未能转换 com.ms.wfc.util.DebugMessageEventHandler
未能转换 com.ms.wfc.util.HandleCollector
未能转换 com.ms.wfc.util.HashTable.addSlot
未能转换 com.ms.wfc.util.HashTable.buckets
未能转换 com.ms.wfc.util.HashTable.findItem
未能转换 com.ms.wfc.util.HashTable.free
未能转换 com.ms.wfc.util.HashTable.getKeys
未能转换 com.ms.wfc.util.HashTable.getValues
未能转换 com.ms.wfc.util.HashTable.growHashTable
未能转换 com.ms.wfc.util.HashTable.HashTable
未能转换 com.ms.wfc.util.HashTable.hashValues
未能转换 com.ms.wfc.util.HashTable.iMax
未能转换 com.ms.wfc.util.HashTable.keys
未能转换 com.ms.wfc.util.HashTable.maxAverageDepth
未能转换 com.ms.wfc.util.HashTable.rehash
未能转换 com.ms.wfc.util.HashTable.removeSlot
未能转换 com.ms.wfc.util.HashTable.removeValue
未能转换 com.ms.wfc.util.HashTable.sizes
未能转换 com.ms.wfc.util.HashTable.values
未能转换 com.ms.wfc.util.IEnumerator.hasMoreItems
未能转换 com.ms.wfc.util.IEnumerator.nextItem
未能转换 com.ms.wfc.util.List.capitalize

未能转换 com.ms.wfc.util.List.clone
未能转换 com.ms.wfc.util.List.getVersion
未能转换 com.ms.wfc.util.List.version
未能转换 com.ms.wfc.util.NumberFormat
未能转换 com.ms.wfc.util.Root.checkLeaks
未能转换 com.ms.wfc.util.Root.get
未能转换 com.ms.wfc.util.StringSorter.compare
未能转换 com.ms.wfc.util.StringSorter.DESCENDING
未能转换 com.ms.wfc.util.StringSorter.sort
未能转换 com.ms.wfc.util.StringSorter.sort(Locale, String, Object, int, int, int)
未能转换 com.ms.wfc.util.Utils.getJavaIdentifier
未能转换 com.ms.wfc.util.Utils.getPrimitiveWord
未能转换 com.ms.wfc.util.Utils.getReservedWord
未能转换 com.ms.wfc.util.Utils.validateIdentifier
未能转换 com.ms.wfc.util.Value.format
未能转换 com.ms.wfc.util.Value.formatCurrency
未能转换 com.ms.wfc.util.Value.formatNumber
未能转换 com.ms.wfc.util.Value.SysFreeString

Com.ms.wfc.win32 错误信息

未能转换 com.ms.wfc.win32.CharBuffer
未能转换 com.ms.wfc.win32.Windows.ACCELERATORHANDLE
未能转换 com.ms.wfc.win32.Windows.BeginPaint
未能转换 com.ms.wfc.win32.Windows.CloseEnhMetaFile
未能转换 com.ms.wfc.win32.Windows.CloseHandle
未能转换 com.ms.wfc.win32.Windows.CopyEnhMetaFile
未能转换 com.ms.wfc.win32.Windows.CopyImage
未能转换 com.ms.wfc.win32.Windows.CreateAcceleratorTable
未能转换 com.ms.wfc.win32.Windows.CreateBitmap
未能转换 com.ms.wfc.win32.Windows.CreateBitmap(int, int, int, int, int[])
未能转换 com.ms.wfc.win32.Windows.CreateBitmap(int, int, int, int, short[])
未能转换 com.ms.wfc.win32.Windows.CreateBrushIndirect
未能转换 com.ms.wfc.win32.Windows.CreateCompatibleBitmap
未能转换 com.ms.wfc.win32.Windows.CreateCompatibleDC
未能转换 com.ms.wfc.win32.Windows.CreateDC
未能转换 com.ms.wfc.win32.Windows.CreateDIBitmap
未能转换 com.ms.wfc.win32.Windows.CreateEllipticRgn
未能转换 com.ms.wfc.win32.Windows.CreateEllipticRgnIndirect
未能转换 com.ms.wfc.win32.Windows.CreateEnhMetaFile
未能转换 com.ms.wfc.win32.Windows.CreateFile
未能转换 com.ms.wfc.win32.Windows.CreateFont
未能转换 com.ms.wfc.win32.Windows.CreateFontIndirect
未能转换 com.ms.wfc.win32.Windows.CreateHalftonePalette
未能转换 com.ms.wfc.win32.Windows.CreateHatchBrush
未能转换 com.ms.wfc.win32.Windows.CreateIC
未能转换 com.ms.wfc.win32.Windows.CreateLockBytesOnHGlobal
未能转换 com.ms.wfc.win32.Windows.CreateMappedBitmap
未能转换 com.ms.wfc.win32.Windows.CreateMenu
未能转换 com.ms.wfc.win32.Windows.CreatePalette
未能转换 com.ms.wfc.win32.Windows.CreatePatternBrush
未能转换 com.ms.wfc.win32.Windows.CreatePen
未能转换 com.ms.wfc.win32.Windows.CreatePenIndirect
未能转换 com.ms.wfc.win32.Windows.CreatePolygonRgn
未能转换 com.ms.wfc.win32.Windows.CreatePolyPolygonRgn
未能转换 com.ms.wfc.win32.Windows.CreatePopupMenu
未能转换 com.ms.wfc.win32.Windows.CreateProcess
未能转换 com.ms.wfc.win32.Windows.CreateRectRgn

未能转换 com.ms.wfc.win32.Windows.CreateRectRgnIndirect
未能转换 com.ms.wfc.win32.Windows.CreateRoundRectRgn
未能转换 com.ms.wfc.win32.Windows.CreateSolidBrush
未能转换 com.ms.wfc.win32.Windows.CreateWindowEx
未能转换 com.ms.wfc.win32.Windows.CURSORHANDLE
未能转换 com.ms.wfc.win32.Windows.DeleteDC
未能转换 com.ms.wfc.win32.Windows.DeleteEnhMetaFile
未能转换 com.ms.wfc.win32.Windows.DeleteObject
未能转换 com.ms.wfc.win32.Windows.DestroyAcceleratorTable
未能转换 com.ms.wfc.win32.Windows.DestroyCursor
未能转换 com.ms.wfc.win32.Windows.DestroyIcon
未能转换 com.ms.wfc.win32.Windows.DestroyMenu
未能转换 com.ms.wfc.win32.Windows.DestroyWindow
未能转换 com.ms.wfc.win32.Windows.DoDragDrop
未能转换 com.ms.wfc.win32.Windows.DrawState
未能转换 com.ms.wfc.win32.Windows.DuplicateHandle
未能转换 com.ms.wfc.win32.Windows.EMFHANDLE
未能转换 com.ms.wfc.win32.Windows.EndPaint
未能转换 com.ms.wfc.win32.Windows.EnumObjects
未能转换 com.ms.wfc.win32.Windows.EnumThreadWindows
未能转换 com.ms.wfc.win32.Windows.FindClose
未能转换 com.ms.wfc.win32.Windows.FindFirstFile
未能转换 com.ms.wfc.win32.Windows.FINDHANDLE
未能转换 com.ms.wfc.win32.Windows.GDIHANDLE
未能转换 com.ms.wfc.win32.Windows.GetDC
未能转换 com.ms.wfc.win32.Windows.GetDCEx
未能转换 com.ms.wfc.win32.Windows.GetEnhMetaFile
未能转换 com.ms.wfc.win32.Windows.GetHGlobalFromILockBytes
未能转换 com.ms.wfc.win32.Windows.GetWindowDC
未能转换 com.ms.wfc.win32.Windows.HDCHANDLE
未能转换 com.ms.wfc.win32.Windows.ICONHANDLE
未能转换 com.ms.wfc.win32.Windows.InvalidateRect
未能转换 com.ms.wfc.win32.Windows.KERNELHANDLE
未能转换 com.ms.wfc.win32.Windows.MENUHANDLE
未能转换 com.ms.wfc.win32.Windows.OleCreatePictureIndirect
未能转换 com.ms.wfc.win32.Windows.PathToRegion
未能转换 com.ms.wfc.win32.Windows.RegisterDragDrop
未能转换 com.ms.wfc.win32.Windows.ReleaseDC
未能转换 com.ms.wfc.win32.Windows.SetEnhMetaFileBits
未能转换 com.ms.wfc.win32.Windows.SetMetaFileBitsEx

未能转换 com.ms.wfc.win32.Windows.SetWindowRgn

未能转换 com.ms.wfc.win32.Windows.SetWinMetaFileBits

未能转换 com.ms.wfc.win32.Windows.StgCreateDocfileOnILockBytes

未能转换 com.ms.wfc.win32.Windows.StgOpenStorageOnILockBytes

未能转换 com.ms.wfc.win32.Windows.WINDOWHANDLE

Com.ms.win32 错误信息

未能转换 com.ms.win32.Kernel32.CreateFiber
未能转换 com.ms.win32.Ole32.CoCreateInstanceEx
未能转换 com.ms.win32.Ole32.CoGetInstanceFromFile
未能转换 com.ms.win32.Ole32.CoGetInstanceFromIStorage
未能转换 com.ms.win32.Ole32.CreateLockBytesOnHGlobal
未能转换 com.ms.win32.Ole32.GetHGlobalFromILockBytes
未能转换 com.ms.win32.Ole32.StgCreateDocfile
未能转换 com.ms.win32.Ole32.StgCreateStorageEx
未能转换 com.ms.win32.Ole32.StgOpenLayoutDocfile
未能转换 com.ms.win32.Ole32.StgOpenStorage
未能转换 com.ms.win32.Spoolss.EnumPrinters

Com.sun.image.codec 错误信息

未能转换 com.sun.image.codec.jpeg.ImageFormatException
未能转换 com.sun.image.codec.jpeg.JPEGCodec
未能转换 com.sun.image.codec.jpeg.JPEGCodec.createJPEGDecoder
未能转换 com.sun.image.codec.jpeg.JPEGCodec.createJPEGEncoder
未能转换 com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(BufferedImage)
未能转换 com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(int,int)
未能转换 com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(JPEGDecodeParam)
未能转换 com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(JPEGEncodeParam)
未能转换 com.sun.image.codec.jpeg.JPEGCodec.getDefaultJPEGEncodeParam(Raster, int)
未能转换 com.sun.image.codec.jpeg.JPEGDecodeParam
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.addDataMarkerData
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setAChuffmanComponentMapping
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setAChuffmanTable
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setDChuffmanComponentMapping
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setDChuffmanTable
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setDensityUnit
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setHorizontalSubsampling
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setImageInfoValid
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setMarkerData
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setQTable
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setQTableComponentMapping
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setQuality
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setRestartInterval
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setTableInfoValid
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setVerticalSubsampling
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setXDensity
未能转换 com.sun.image.codec.jpeg.JPEGEncodeParam.setYDensity
未能转换 com.sun.image.codec.jpeg.JPEGHuffmanTable
未能转换 com.sun.image.codec.jpeg.JPEGImageDecoder
未能转换 com.sun.image.codec.jpeg.JPEGImageDecoder.decodeAsBufferedImage
未能转换 com.sun.image.codec.jpeg.JPEGImageDecoder.decodeAsRaster
未能转换 com.sun.image.codec.jpeg.JPEGImageDecoder.getInputStream
未能转换 com.sun.image.codec.jpeg.JPEGImageDecoder.getJPEGDecodeParam
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.encode(BufferedImage)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.encode(BufferedImage, JPEGEncodeParam)

未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.encode(Raster)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.encode(Raster, JPEGEncodeParam)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultColorId
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGDecodeParam)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, BufferedImage)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, int, int)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getDefaultJPEGEncodeParam(JPEGEncodeParam, Raster,int)
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getJPEGEncodeParam
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.getOutputStream
未能转换 com.sun.image.codec.jpeg.JPEGImageEncoder.setJPEGEncodeParam
未能转换 com.sun.image.codec.jpeg.JPEGQTable
未能转换 com.sun.image.codec.jpeg.TruncatedFileException

Java.applet 错误信息

未能转换 java.applet.Applet
未能转换 java.applet.Applet.destroy
未能转换 java.applet.Applet.getAccessibleContext
未能转换 java.applet.Applet.getAppletContext
未能转换 java.applet.Applet getAppletInfo
未能转换 java.applet.Applet.getAudioClip
未能转换 java.applet.Applet.getCodeBase
未能转换 java.applet.Applet.getDocumentBase
未能转换 java.applet.Applet.getImage
未能转换 java.applet.Applet.getLocale
未能转换 java.applet.Applet.getParameter
未能转换 java.applet.Applet.getParameterInfo
未能转换 java.applet.Applet.init
未能转换 java.applet.Applet.isActive
未能转换 java.applet.Applet.newAudioClip
未能转换 java.applet.Applet.play
未能转换 java.applet.Applet.resize
未能转换 java.applet.Applet.setStub
未能转换 java.applet.Applet.showStatus
未能转换 java.applet.Applet.start
未能转换 java.applet.Applet.stop
未能转换 java.applet.AppletContext
未能转换 java.applet.AppletStub
未能转换 java.applet.AudioClip

Java.awt 错误信息

未能转换 java.awt.<ClassName>.add*Listener
未能转换 java.awt.<ClassName>.addNotify
未能转换 java.awt.<ClassName>.getPeer
未能转换 java.awt.<ClassName>.print*
未能转换 java.awt.<ClassName>.processActionEvent
未能转换 java.awt.<ClassName>.processEvent
未能转换 java.awt.<ClassName>.processItemEvent
未能转换 java.awt.<ClassName>.remove*Listener
未能转换 java.awt.<ClassName>.removeNotify
未能转换 java.awt.ActiveEvent
未能转换 java.awt.Adjustable
未能转换 java.awt.AlphaComposite
未能转换 java.awt.AWTError
未能转换 java.awt.AWTEvent
未能转换 java.awt.AWTEvent.<EventType>
未能转换 java.awt.AWTEvent.AWTEvent
未能转换 java.awt.AWTEvent.consume
未能转换 java.awt.AWTEvent.consumed
未能转换 java.awt.AWTEvent.getID
未能转换 java.awt.AWTEvent.id
未能转换 java.awt.AWTEvent.isConsumed
未能转换 java.awt.AWTEventMulticaster
未能转换 java.awt.AWTPermission.AWTPermission
未能转换 java.awt.BasicStroke.BasicStroke
未能转换 java.awt.BasicStroke.createStrokedShape
未能转换 java.awt.BorderLayout
未能转换 java.awt.Button.getActionCommand
未能转换 java.awt.Button.getListeners
未能转换 java.awt.Button.setActionCommand
未能转换 java.awt.Canvas
未能转换 java.awt.Canvas.Canvas
未能转换 java.awt.Canvas.paint
未能转换 java.awt.CardLayout
未能转换 java.awt.Checkbox.getListeners
未能转换 java.awt.CheckboxGroup.getCurrent
未能转换 java.awt.CheckboxGroup.getSelectedCheckbox
未能转换 java.awt.CheckboxMenuItem.getAccessibleContext

未能转换 java.awt.CheckboxMenuItem.getListeners
未能转换 java.awt.Choice
未能转换 java.awt.Choice.getListeners
未能转换 java.awt.Choice.getSelectedIndex
未能转换 java.awt.Choice.getSelectedItem
未能转换 java.awt.Color(ColorSpace, float[], float)
未能转换 java.awt.Color(Color(float, float, float, float))
未能转换 java.awt.Color(Color(int))
未能转换 java.awt.Color(Color(int, boolean))
未能转换 java.awt.Color(Color(int, int, int,int))
未能转换 java.awt.Color.createContext
未能转换 java.awt.Color.decode
未能转换 java.awt.Color.getColor
未能转换 java.awt.Color.getColorComponents(ColorSpace, float[])
未能转换 java.awt.Color.getColorComponents(float[])
未能转换 java.awt.Color.getColorSpace
未能转换 java.awt.Color.getComponents(ColorSpace, float[])
未能转换 java.awt.Color.getComponents(float[])
未能转换 java.awt.Color.getHSBColor
未能转换 java.awt.Color.getRGBColorComponents
未能转换 java.awt.Color.getRGBComponents
未能转换 java.awt.Color.getTransparency
未能转换 java.awt.Color.HSBtoRGB
未能转换 java.awt.Color.RGBtoHSB
未能转换 java.awt.Component.action
未能转换 java.awt.Component.addHierarchyBoundsListener
未能转换 java.awt.Component.addHierarchyListener
未能转换 java.awt.Component.addInputMethodListener
未能转换 java.awt.Component.addPropertyChangeListener
未能转换 java.awt.Component.BOTTOM_ALIGNMENT
未能转换 java.awt.Component.CENTER_ALIGNMENT
未能转换 java.awt.Component.checkImage
未能转换 java.awt.Component.coalesceEvents
未能转换 java.awt.Component.createImage(ImageProducer)
未能转换 java.awt.Component.createImage(int, int)
未能转换 java.awt.Component.deliverEvent
未能转换 java.awt.Component.disable
未能转换 java.awt.Component.disableEvents
未能转换 java.awt.Component.dispatchEvent
未能转换 java.awt.Component.enableEvents

未能转换 java.awt.Component.enableInputMethods
未能转换 java.awt.Component.firePropertyChange
未能转换 java.awt.Component.getAlignmentX
未能转换 java.awt.Component.getAlignmentY
未能转换 java.awt.Component.getColorModel
未能转换 java.awt.Component.getDropTarget
未能转换 java.awt.Component.getGraphicsConfiguration
未能转换 java.awt.Component.getInputContext
未能转换 java.awt.Component.getInputMethodRequests
未能转换 java.awt.Component.getListeners
未能转换 java.awt.Component.getLocale
未能转换 java.awt.Component.getMaximumSize
未能转换 java.awt.Component.getMinimumSize
未能转换 java.awt.Component.getPreferredSize
未能转换 java.awt.Component.getSize
未能转换 java.awt.Component.getToolkit
未能转换 java.awt.Component.getTreeLock
未能转换 java.awt.Component.handleEvent
未能转换 java.awt.Component.hasFocus
未能转换 java.awt.Component.imageUpdate
未能转换 java.awt.Component.isDisplayable
未能转换 java.awt.Component.isDoubleBuffered
未能转换 java.awt.Component.isLightweight
未能转换 java.awt.Component.isOpaque
未能转换 java.awt.Component.isValid
未能转换 java.awt.Component.isVisible
未能转换 java.awt.Component.keyDown
未能转换 java.awt.Component.keyUp
未能转换 java.awt.Component.LEFT_ALIGNMENT
未能转换 java.awt.Component.lostFocus
未能转换 java.awt.Component.minimumSize
未能转换 java.awt.Component.mouseDown
未能转换 java.awt.Component.mouseDrag
未能转换 java.awt.Component.mouseEnter
未能转换 java.awt.Component.mouseExit
未能转换 java.awt.Component.mouseMove
未能转换 java.awt.Component.mouseUp
未能转换 java.awt.Component.move
未能转换 java.awt.Component.nextFocus
未能转换 java.awt.Component.paint

未能转换 java.awt.Component.paintAll
未能转换 java.awt.Component.postEvent
未能转换 java.awt.Component.preferredSize
未能转换 java.awt.Component.prepareImage
未能转换 java.awt.Component.process*Event
未能转换 java.awt.Component.remove
未能转换 java.awt.Component.removeInputMethodListener
未能转换 java.awt.Component.removePropertyChangeListener
未能转换 java.awt.Component.repaint
未能转换 java.awt.Component.repaint(int, int, int, int)
未能转换 java.awt.Component.repaint(long)
未能转换 java.awt.Component.repaint(long, int, int, int, int)
未能转换 java.awt.Component.RIGHT_ALIGNMENT
未能转换 java.awt.Component.setBounds
未能转换 java.awt.Component.setComponentOrientation
未能转换 java.awt.Component.setDropTarget
未能转换 java.awt.Component.setEnabled
未能转换 java.awt.Component.setLocale
未能转换 java.awt.Component.setLocation
未能转换 java.awt.Component.setSize
未能转换 java.awt.Component.setVisible
未能转换 java.awt.Component.TOP_ALIGNMENT
未能转换 java.awt.Component.transferFocus
未能转换 java.awt.Component.validate
未能转换 java.awt.ComponentOrientation.getOrientation(Locale)
未能转换 java.awt.ComponentOrientation.getOrientation(ResourceBundle)
未能转换 java.awt.ComponentOrientation.isHorizontal
未能转换 java.awt.Composite
未能转换 java.awt.CompositeContext
未能转换 java.awt.Container.add(Component)
未能转换 java.awt.Container.add(Component, int)
未能转换 java.awt.Container.add(Component, Object)
未能转换 java.awt.Container.add(Component, Object, int)
未能转换 java.awt.Container.add(String, Component)
未能转换 java.awt.Container.deliverEvent
未能转换 java.awt.Container.getComponent
未能转换 java.awt.Container.getComponentAt(int, int)
未能转换 java.awt.Container.getComponentAt(Point)
未能转换 java.awt.Container.getLayout
未能转换 java.awt.Container.getListeners

未能转换 java.awt.Container.paint*

未能转换 java.awt.Container.processContainerEvent

未能转换 java.awt.Container.setLayout

未能转换 java.awt.Container.validate

未能转换 java.awt.Container.validateTree

未能转换 java.awt.Cursor.Cursor

未能转换 java.awt.Cursor.CUSTOM_CURSOR

未能转换 java.awt.Cursor.getName

未能转换 java.awt.Cursor.getSystemCustomCursor

未能转换 java.awt.Cursor.name

未能转换 java.awt.Cursor.predefined

未能转换 java.awt.Dialog.Dialog

未能转换 java.awt.Dialog.hide

未能转换 java.awt.Dialog.setModal

未能转换 java.awt.Dialog.show

未能转换 java.awt.Dimension.setSize

未能转换 java.awt.Dimension.toString

未能转换 java.awt.Event

未能转换 java.awt.EventQueue

未能转换 java.awt.FileDialog.*FilenameFilter

未能转换 java.awt.FileDialog.FileDialog(Frame)

未能转换 java.awt.FileDialog.FileDialog(Frame, String)

未能转换 java.awt.FileDialog.FileDialog(Frame, String, int)

未能转换 java.awt.FileDialog.getFile

未能转换 java.awt.FileDialog.setMode

未能转换 java.awt.FlowLayout

未能转换 java.awtFlowLayout.addComponent

未能转换 java.awtFlowLayout.FlowLayout

未能转换 java.awtFlowLayout.FlowLayout(int)

未能转换 java.awtFlowLayout.FlowLayout(int, int, int)

未能转换 java.awtFlowLayout.getAlignment

未能转换 java.awtFlowLayout.getHgap

未能转换 java.awtFlowLayout.getVgap

未能转换 java.awtFlowLayout.layoutContainer

未能转换 java.awtFlowLayout.minimumLayoutSize

未能转换 java.awtFlowLayout.preferredLayoutSize

未能转换 java.awtFlowLayout.removeLayoutComponent

未能转换 java.awtFlowLayout.setAlignment

未能转换 java.awtFlowLayout.setHgap

未能转换 java.awtFlowLayout.setVgap

未能转换 java.awt.FlowLayout.toString
未能转换 java.awt.Font.canDisplay
未能转换 java.awt.Font.canDisplayUpTo
未能转换 java.awt.Font.CENTER_BASELINE
未能转换 java.awt.Font.createFont
未能转换 java.awt.Font.createGlyphVector
未能转换 java.awt.Font.deriveFont(AffineTransform)
未能转换 java.awt.Font.deriveFont(int, AffineTransform)
未能转换 java.awt.Font.deriveFont(Map)
未能转换 java.awt.Font.Font
未能转换 java.awt.Font.getAttributes
未能转换 java.awt.Font.getAvailableAttributes
未能转换 java.awt.Font.getBaselineFor
未能转换 java.awt.Font.getFont
未能转换 java.awt.Font.getFont(Map)
未能转换 java.awt.Font.getItalicAngle
未能转换 java.awt.Font.getLineMetrics
未能转换 java.awt.Font.getLineMetrics(CharacterIterator, int, int, FontRenderContext)
未能转换 java.awt.Font.getMaxCharBounds
未能转换 java.awt.Font.getMissingGlyphCode
未能转换 java.awt.Font.getNumGlyphs
未能转换 java.awt.Font.getPSName
未能转换 java.awt.Font.getStringBounds(char[], int, int, FontRenderContext)
未能转换 java.awt.Font.getStringBounds(CharacterIterator, int, int, FontRenderContext)
未能转换 java.awt.Font.getStringBounds(String, FontRenderContext)
未能转换 java.awt.Font.getStringBounds(String, int, int, FontRenderContext)
未能转换 java.awt.Font.getTransform
未能转换 java.awt.Font.HANGING_BASELINE
未能转换 java.awt.Font.hasUniformLineMetrics
未能转换 java.awt.Font.PLAIN
未能转换 java.awt.Font.ROMAN_BASELINE
未能转换 java.awt.Font.TRUETYPE_FONT
未能转换 java.awt.FontMetrics.*Width
未能转换 java.awt.FontMetrics.FontMetrics
未能转换 java.awt.FontMetrics.getLeading
未能转换 java.awt.FontMetrics.getLineMetrics
未能转换 java.awt.FontMetrics.getMaxAdvance
未能转换 java.awt.FontMetrics.getMaxCharBounds
未能转换 java.awt.FontMetrics.getStringBounds(char[], int, int, Graphics)
未能转换 java.awt.FontMetrics.getStringBounds(CharacterIterator, int, int, Graphics)

未能转换 java.awt.FontMetrics.getStringBounds(String, Graphics)
未能转换 java.awt.FontMetrics.getStringBounds(String, int, int, Graphics)
未能转换 java.awt.FontMetrics.getWidths
未能转换 java.awt.FontMetrics.hasUniformLineMetrics
未能转换 java.awt.Frame
未能转换 java.awt.Frame.Frame(GraphicsConfiguration)
未能转换 java.awt.Frame.Frame(String, GraphicsConfiguration)
未能转换 java.awt.Frame.getCursorType
未能转换 java.awt.Frame.getFrames
未能转换 java.awt.Frame.getIconImage
未能转换 java.awt.Frame.setIconImage
未能转换 java.awt.Frame.setState
未能转换 java.awt.GradientPaint.createContext
未能转换 java.awt.GradientPaint.getTransparency
未能转换 java.awt.GradientPaint.GradientPaint
未能转换 java.awt.GradientPaint.isCyclic
未能转换 java.awt.Graphics.copyArea
未能转换 java.awt.Graphics.drawBytes
未能转换 java.awt.Graphics.drawChars
未能转换 java.awt.Graphics.drawRoundRect
未能转换 java.awt.Graphics.drawString
未能转换 java.awt.Graphics.drawString(AttributedCharacterIterator, int, int)
未能转换 java.awt.Graphics.drawString(int, int)
未能转换 java.awt.Graphics.fill3DRect
未能转换 java.awt.Graphics.fillRoundRect
未能转换 java.awt.Graphics.finalize
未能转换 java.awt.Graphics.getClip
未能转换 java.awt.Graphics.setClip
未能转换 java.awt.Graphics.setPaintMode
未能转换 java.awt.Graphics.setXORMode
未能转换 java.awt.Graphics2D.addRenderingHints
未能转换 java.awt.Graphics2D.clip
未能转换 java.awt.Graphics2D.draw
未能转换 java.awt.Graphics2D.drawGlyphVector
未能转换 java.awt.Graphics2D.drawImage
未能转换 java.awt.Graphics2D.drawRenderableImage
未能转换 java.awt.Graphics2D.drawRenderedImage
未能转换 java.awt.Graphics2D.drawString
未能转换 java.awt.Graphics2D.fill
未能转换 java.awt.Graphics2D.fill3DRect

未能转换 java.awt.Graphics2D.getComposite
未能转换 java.awt.Graphics2D.getDeviceConfiguration
未能转换 java.awt.Graphics2D.getPaint
未能转换 java.awt.Graphics2D.getRenderingHint
未能转换 java.awt.Graphics2D.getRenderingHints
未能转换 java.awt.Graphics2D.getStroke
未能转换 java.awt.Graphics2D.Graphics2D
未能转换 java.awt.Graphics2D.hit
未能转换 java.awt.Graphics2D.rotate
未能转换 java.awt.Graphics2D.setComposite
未能转换 java.awt.Graphics2D.setPaint
未能转换 java.awt.Graphics2D.setRenderingHint
未能转换 java.awt.Graphics2D.setRenderingHints
未能转换 java.awt.Graphics2D.setStroke
未能转换 java.awt.Graphics2D.shear
未能转换 java.awt.Graphics2D.transform
未能转换 java.awt.GraphicsConfigTemplate
未能转换 java.awt.GraphicsConfiguration
未能转换 java.awt.GraphicsDevice
未能转换 java.awt.GraphicsEnvironment
未能转换 java.awt.GraphicsEnvironment.getAllFonts
未能转换 java.awt.GraphicsEnvironment.getAvailableFontFamilyNames
未能转换 java.awt.GraphicsEnvironment.getAvailableFontFamilyNames(Locale)
未能转换 java.awt.GraphicsEnvironment.getDefaultScreenDevice
未能转换 java.awt.GraphicsEnvironment.getScreenDevices
未能转换 java.awt.GridBagConstraints
未能转换 java.awt.GridBagLayout
未能转换 java.awt.GridLayout
未能转换 java.awt.GridLayout.addComponent
未能转换 java.awt.GridLayout.GridLayout
未能转换 java.awt.GridLayout.GridLayout(int, int)
未能转换 java.awt.GridLayout.GridLayout(int, int, int, int)
未能转换 java.awt.GridLayout.layoutContainer
未能转换 java.awt.GridLayout.minimumLayoutSize
未能转换 java.awt.GridLayout.preferredLayoutSize
未能转换 java.awt.GridLayout.removeLayoutComponent
未能转换 java.awt.Image.getProperty
未能转换 java.awt.Image.SCALE_<Algorithm>
未能转换 java.awt.Image.UndefinedProperty
未能转换 java.awt.ItemSelectable

未能转换 java.awt.JobAttributes.DestinationType
未能转换 java.awt.JobAttributes.DialogType
未能转换 java.awt.JobAttributes.getDefaultSelection
未能转换 java.awt.JobAttributes.getDestination
未能转换 java.awt.JobAttributes.getDialog
未能转换 java.awt.JobAttributes.getFileName
未能转换 java.awt.JobAttributes.getMultipleDocumentHandling
未能转换 java.awt.JobAttributes.getPageRanges
未能转换 java.awt.JobAttributes.getSides
未能转换 java.awt.JobAttributes.JobAttributes
未能转换 java.awt.JobAttributes.MultipleDocumentHandlingType
未能转换 java.awt.JobAttributes.setDefaultSelection
未能转换 java.awt.JobAttributes.setDestination
未能转换 java.awt.JobAttributes.setDialog
未能转换 java.awt.JobAttributes.setFileName
未能转换 java.awt.JobAttributes.setMultipleDocumentHandling
未能转换 java.awt.JobAttributes.setMultipleDocumentHandlingToDefault
未能转换 java.awt.JobAttributes setPageRanges
未能转换 java.awt.JobAttributes.setSides
未能转换 java.awt.JobAttributes.setSidesToDefault
未能转换 java.awt.Label.setAlignment
未能转换 java.awt.LayoutManager
未能转换 java.awt.LayoutManager2
未能转换 java.awt.List.delItems
未能转换 java.awt.List.getListeners
未能转换 java.awt.MediaTracker
未能转换 java.awt.Menu.getAccessibleContext
未能转换 java.awt.Menu.isTearOff
未能转换 java.awt.MenuBar.deleteShortcut
未能转换 java.awt.MenuBar.getAccessibleContext
未能转换 java.awt.MenuBar.getHelpMenu
未能转换 java.awt.MenuBar.getShortcutMenuItem
未能转换 java.awt.MenuBar.setHelpMenu
未能转换 java.awt.MenuBar.shortcuts
未能转换 java.awt.MenuComponent.dispatchEvent
未能转换 java.awt.MenuComponent.getAccessibleContext
未能转换 java.awt.MenuComponent.getTreeLock
未能转换 java.awt.MenuComponent.MenuComponent
未能转换 java.awt.MenuComponent.postEvent
未能转换 java.awt.MenuComponent.setFont

未能转换 java.awt.MenuContainer.postEvent
未能转换 java.awt.MenuItem.disableEvents
未能转换 java.awt.MenuItem.enableEvents
未能转换 java.awt.MenuItem.getAccessibleContext
未能转换 java.awt.MenuItem.getActionCommand
未能转换 java.awt.MenuItem.getListeners
未能转换 java.awt.MenuItem.setActionCommand
未能转换 java.awt.MenuShortcut.MenuShortcut
未能转换 java.awt.MenuShortcut.usesShiftModifier
未能转换 java.awt.PageAttributes.getOrigin
未能转换 java.awt.PageAttributes.MediaType.<PaperKind>
未能转换 java.awt.PageAttributes.PageAttributes
未能转换 java.awt.PageAttributes.setMedia
未能转换 java.awt.PageAttributes.setMediaToDefault
未能转换 java.awt.PageAttributes.setOrigin
未能转换 java.awt.PageAttributes.setPrinterResolution(int)
未能转换 java.awt.PageAttributes.setPrinterResolution(int[])
未能转换 java.awt.PageAttributes.setPrintQuality(int)
未能转换 java.awt.PageAttributes.setPrintQuality(PageAttributesPrintQualityType)
未能转换 java.awt.Paint.createContext
未能转换 java.awt.PaintContext
未能转换 java.awt.Panel.Panel
未能转换 java.awt.Point.getX
未能转换 java.awt.Point.getY
未能转换 java.awt.Point.setLocation
未能转换 java.awt.Polygon.addPoint
未能转换 java.awt.Polygon.bounds
未能转换 java.awt.Polygon.contains(double, double)
未能转换 java.awt.Polygon.contains(double, double, double, double)
未能转换 java.awt.Polygon.contains(Rectangle2D)
未能转换 java.awt.Polygon.getPathIterator
未能转换 java.awt.Polygon.intersects(double, double, double, double)
未能转换 java.awt.Polygon.intersects(Rectangle2D)
未能转换 java.awt.Polygon.npoints
未能转换 java.awt.Polygon.Polygon
未能转换 java.awt.PopupMenu.getAccessibleContext
未能转换 java.awt.PopupMenu.show
未能转换 java.awt.PrintGraphics
未能转换 java.awt.PrintJob
未能转换 java.awt.Rectangle.outcode

未能转换 java.awt.Rectangle.setRect
未能转换 java.awt.RenderingHints.<PropertySetting>
未能转换 java.awt.RenderingHints.add
未能转换 java.awt.RenderingHints.Key
未能转换 java.awt.RenderingHints.putAll
未能转换 java.awt.RenderingHints.RenderingHints
未能转换 java.awt.Robot
未能转换 java.awt.Scrollbar
未能转换 java.awt.Scrollbar.addAdjustmentListener
未能转换 java.awt.Scrollbar.getListeners
未能转换 java.awt.Scrollbar.getVisible
未能转换 java.awt.Scrollbar.getVisibleAmount
未能转换 java.awt.Scrollbar paramString
未能转换 java.awt.Scrollbar.processAdjustmentEvent
未能转换 java.awt.Scrollbar.Scrollbar
未能转换 java.awt.Scrollbar.Scrollbar
未能转换 java.awt.Scrollbar.setOrientation
未能转换 java.awt.Scrollbar.setVisibleAmount
未能转换 java.awt.ScrollPane.get*Adjustable
未能转换 java.awt.ScrollPane.printComponents
未能转换 java.awt.ScrollPane.ScrollPane
未能转换 java.awt.ScrollPane.setLayout
未能转换 java.awt.Shape
未能转换 java.awt.Shape.contains(double,double)
未能转换 java.awt.Shape.contains(double, double, double, double)
未能转换 java.awt.Shape.contains(Rectangle2D)
未能转换 java.awt.Shape.getPathIterator
未能转换 java.awt.Shape.intersects(double,double,double,double)
未能转换 java.awt.Shape.intersects(Rectangle2D)
未能转换 java.awt.Stroke
未能转换 java.awt.Stroke.createStrokedShape
未能转换 java.awt.SystemColor.createContext
未能转换 java.awt.SystemColor.NUM_COLORS
未能转换 java.awt.SystemColor.TEXT
未能转换 java.awt.SystemColor.TEXT_HIGHLIGHT
未能转换 java.awt.SystemColor.TEXT_HIGHLIGHT_TEXT
未能转换 java.awt.SystemColor.TEXT_INACTIVE_TEXT
未能转换 java.awt.SystemColor.TEXT_TEXT
未能转换 java.awt.SystemColor.textHighlight
未能转换 java.awt.SystemColor.textHighlightText

未能转换 java.awt.SystemColor.textInactiveText
未能转换 java.awt.SystemColor.textText
未能转换 java.awt.TextArea.*Columns
未能转换 java.awt.TextArea.*Rows
未能转换 java.awt.TextComponent.enableInputMethods
未能转换 java.awt.TextComponent.getBackground
未能转换 java.awt.TextComponent.getListeners
未能转换 java.awt.TextComponent.processTextEvent
未能转换 java.awt.TextComponent.setText
未能转换 java.awt.TextComponent.textListener
未能转换 java.awt.TextField.*Columns
未能转换 java.awt.TextField.setColumns
未能转换 java.awt.TextField.getListeners
未能转换 java.awt.TextField.getPreferredSize
未能转换 java.awt.TextField.preferredSize
未能转换 java.awt.TextField.setColumns
未能转换 java.awt.TextField.TextField
未能转换 java.awt.TexturePaint.createContext
未能转换 java.awt.TexturePaint.getAnchorRect
未能转换 java.awt.TexturePaint.getTransparency
未能转换 java.awt.Toolkit
未能转换 java.awt.Transparency
未能转换 java.awt.Window.apply ResourceBundle
未能转换 java.awt.Window.getGraphicsConfiguration
未能转换 java.awt.Window.getInputContext
未能转换 java.awt.Window.getListeners
未能转换 java.awt.Window.getToolkit
未能转换 java.awt.Window.getWarningString
未能转换 java.awt.Window.pack
未能转换 java.awt.Window.postEvent
未能转换 java.awt.Window.processWindowEvent
未能转换 java.awt.Window.Window

Java.awt.color 错误信息

未能转换 java.awt.color.ColorSpace

未能转换 java.awt.color.ICC_ColorSpace

未能转换 java.awt.color.ICC_Profile

未能转换 java.awt.color.ICC_ProfileGray

未能转换 java.awt.color.ICC_ProfileRGB

Java.awt.datatransfer 错误信息

未能转换 java.awt.datatransfer.Clipboard.Clipboard
未能转换 java.awt.datatransfer.Clipboard.contents
未能转换 java.awt.datatransfer.Clipboard.getName
未能转换 java.awt.datatransfer.Clipboard.owner
未能转换 java.awt.datatransfer.ClipboardOwner
未能转换 java.awt.datatransfer.DataFlavor.clone
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(Class, String)
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(DataFlavor)
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(String)
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(String, String)
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(String, String, ClassLoader)
未能转换 java.awt.datatransfer.DataFlavor.DataFlavor(String, String, MimeTypeParameterList, class, j)
未能转换 java.awt.datatransfer.DataFlavor.equals
未能转换 java.awt.datatransfer.DataFlavor.getDefaultRepresentationClass
未能转换 java.awt.datatransfer.DataFlavor.getDefaultRepresentationClassAsString
未能转换 java.awt.datatransfer.DataFlavor.getHumanPresentableName
未能转换 java.awt.datatransfer.DataFlavor.getMimeType
未能转换 java.awt.datatransfer.DataFlavor.getParameter
未能转换 java.awt.datatransfer.DataFlavor.getPrimaryType
未能转换 java.awt.datatransfer.DataFlavor.getReaderForText
未能转换 java.awt.datatransfer.DataFlavor.getRepresentationClass
未能转换 java.awt.datatransfer.DataFlavor.getSubType
未能转换 java.awt.datatransfer.DataFlavor.isFlavorJavaFileType
未能转换 java.awt.datatransfer.DataFlavor.isFlavorRemoteObjectType
未能转换 java.awt.datatransfer.DataFlavor.isMimeTypeEqual
未能转换 java.awt.datatransfer.DataFlavor.isMimeTypeSerializedObject
未能转换 java.awt.datatransfer.DataFlavor.isRepresentationClassInputStream
未能转换 java.awt.datatransfer.DataFlavor.isRepresentationClassRemote
未能转换 java.awt.datatransfer.DataFlavor.isRepresentationClassSerializable
未能转换 java.awt.datatransfer.DataFlavor.javaFileDialogType
未能转换 java.awt.datatransfer.DataFlavor.javaJVMLocalObjectMimeType
未能转换 java.awt.datatransfer.DataFlavor.javaRemoteObjectMimeType
未能转换 java.awt.datatransfer.DataFlavor.normalizeMimeType
未能转换 java.awt.datatransfer.DataFlavor.normalizeMimeTypeParameter
未能转换 java.awt.datatransfer.DataFlavor.readExternal
未能转换 java.awt.datatransfer.DataFlavor.selectBestTextFlavor

未能转换 java.awt.datatransfer.DataFlavor.setHumanPresentableName
未能转换 java.awt.datatransfer.DataFlavor.tryToLoadClass
未能转换 java.awt.datatransfer.DataFlavor.writeExternal
未能转换 java.awt.datatransfer.FlavorMap
未能转换 java.awt.datatransfer.StringSelection.getTransferDataFlavors
未能转换 java.awt.datatransfer.StringSelection.isDataFlavorSupported
未能转换 java.awt.datatransfer.StringSelection.lostOwnership
未能转换 java.awt.datatransfer.SystemFlavorMap
未能转换 java.awt.datatransfer.Transferable.getTransferDataFlavors
未能转换 java.awt.datatransfer.Transferable.isDataFlavorSupported

Java.awt.dnd 错误信息

未能转换 java.awt.dnd.Autoscroll
未能转换 java.awt.dnd.DragGestureEvent.DragGestureEvent
未能转换 java.awt.dnd.DragGestureEvent.getDragAction
未能转换 java.awt.dnd.DragGestureEvent.getDragSource
未能转换 java.awt.dnd.DragGestureEvent.getSourceAsDragGestureRecognizer
未能转换 java.awt.dnd.DragGestureEvent.getTriggerEvent
未能转换 java.awt.dnd.DragGestureEvent.iterator
未能转换 java.awt.dnd.DragGestureEvent.toArray
未能转换 java.awt.dnd.DragGestureRecognizer
未能转换 java.awt.dnd.DragSource.createDragSourceContext
未能转换 java.awt.dnd.DragSource.DefaultCopyDrop
未能转换 java.awt.dnd.DragSource.DefaultCopyNoDrop
未能转换 java.awt.dnd.DragSource.DefaultLinkDrop
未能转换 java.awt.dnd.DragSource.DefaultLinkNoDrop
未能转换 java.awt.dnd.DragSource.DefaultMoveDrop
未能转换 java.awt.dnd.DragSource.DefaultMoveNoDrop
未能转换 java.awt.dnd.DragSource.DragSource
未能转换 java.awt.dnd.DragSource.getDefaultDragSource
未能转换 java.awt.dnd.DragSource.getFlavorMap
未能转换 java.awt.dnd.DragSource.isDragImageSupported
未能转换 java.awt.dnd.DragSourceContext
未能转换 java.awt.dnd.DragSourceDragEvent.DragSourceDragEvent
未能转换 java.awt.dnd.DragSourceDropEvent.DragSourceDropEvent
未能转换 java.awt.dnd.DragSourceDropEvent.getDropSuccess
未能转换 java.awt.dnd.DragSourceEvent.DragSourceEvent
未能转换 java.awt.dnd.DragSourceEvent.getDragSourceContext
未能转换 java.awt.dnd.DragSourceListener.dropActionChanged
未能转换 java.awt.dnd.DropTarget
未能转换 java.awt.dnd.DropTarget.setComponent
未能转换 java.awt.dnd.DropTargetContext
未能转换 java.awt.dnd.DropTargetDragEvent.DropTargetDragEvent
未能转换 java.awt.dnd.DropTargetDragEvent.getCurrentDataFlavors
未能转换 java.awt.dnd.DropTargetDragEvent.getCurrentDataFlavorsAsList
未能转换 java.awt.dnd.DropTargetDragEvent.isDataFlavorSupported
未能转换 java.awt.dnd.DropTargetDragEvent.rejectDrag
未能转换 java.awt.dnd.DropTargetDropEvent.dropComplete
未能转换 java.awt.dnd.DropTargetDropEvent.DropTargetDropEvent(DropTargetContext, Point, int, int)

未能转换 java.awt.dnd.DropTargetDropEvent.DropTargetDropEvent(DropTargetContext, Point, int, int, boolean)
未能转换 java.awt.dnd.DropTargetDropEvent.getCurrentDataFlavors
未能转换 java.awt.dnd.DropTargetDropEvent.getCurrentDataFlavorsAsList
未能转换 java.awt.dnd.DropTargetDropEvent.isDataFlavorSupported
未能转换 java.awt.dnd.DropTargetDropEvent.isLocalTransfer
未能转换 java.awt.dnd.DropTargetDropEvent.rejectDrop
未能转换 java.awt.dnd.DropTargetEvent
未能转换 java.awt.dnd.InvalidDnDOperationException
未能转换 java.awt.dnd.MouseDragGestureRecognizer
未能转换 java.awt.dnd.peer.DragSourceContextPeer
未能转换 java.awt.dnd.peer.DropTargetContextPeer
未能转换 java.awt.dnd.peer.DropTargetPeer

Java.awt.event 错误信息

未能转换 java.awt.event.<ClassName>.<EventType>_FIRST
未能转换 java.awt.event.<ClassName>.<EventType>_LAST
未能转换 java.awt.event.ActionEvent.<KeyName>_MASK
未能转换 java.awt.event.ActionEvent.ACTION_PERFORMED
未能转换 java.awt.event.ActionEvent.getActionCommand
未能转换 java.awt.event.AWTEventListener
未能转换 java.awt.event.ComponentEvent.COMPONENT_<EventType>
未能转换 java.awt.event.FocusEvent.FOCUS_<EventType>
未能转换 java.awt.event.FocusEvent.isTemporary
未能转换 java.awt.event.HierarchyBoundsAdapter
未能转换 java.awt.event.HierarchyBoundsListener
未能转换 java.awt.event.HierarchyEvent
未能转换 java.awt.event.HierarchyListener
未能转换 java.awt.event.InputEvent
未能转换 java.awt.event.InputMethodEvent
未能转换 java.awt.event.InputMethodListener
未能转换 java.awt.event.InvocationEvent
未能转换 java.awt.event.ItemEvent.<EventType>
未能转换 java.awt.event.ItemEvent.getStateChange
未能转换 java.awt.event.KeyEvent.<VirtualKey>
未能转换 java.awt.event.KeyEvent.CHAR_UNDEFINED
未能转换 java.awt.event.KeyEvent.getKeyChar
未能转换 java.awt.event.KeyEvent.getKeyCode
未能转换 java.awt.event.KeyEvent.getKeyModifiersText
未能转换 java.awt.event.KeyEvent.getKeyText
未能转换 java.awt.event.KeyEvent.isActionKey
未能转换 java.awt.event.KeyEvent.KEY_<EventType>
未能转换 java.awt.event.KeyEvent.setKeyChar
未能转换 java.awt.event.KeyEvent.setKeyCode
未能转换 java.awt.event.KeyEvent.setModifiers
未能转换 java.awt.event.KeyEvent.setSource
未能转换 java.awt.event.KeyEvent.VK.<VirtualKey>
未能转换 java.awt.event.KeyEvent.VK_<CharacterName>
未能转换 java.awt.event.MouseEvent.getClickCount
未能转换 java.awt.event.MouseEvent.getPoint
未能转换 java.awt.event.MouseEvent.getX
未能转换 java.awt.event.MouseEvent.getY

未能转换 java.awt.event.MouseEvent.isPopupTrigger
未能转换 java.awt.event.MouseEvent.MOUSE_<EventType>
未能转换 java.awt.event.MouseEvent.translatePoint
未能转换 java.awt.event.PaintEvent.<EventType>
未能转换 java.awt.event.PaintEvent.setUpdateRect
未能转换 java.awt.event.TextEvent.TEXT_VALUE_CHANGED
未能转换 java.awt.event.WindowEvent.WINDOW_<EventType>

Java.awt.font 错误信息

未能转换 java.awt.FontRenderContext
未能转换 java.awt.FontRenderContext.FontRenderContext
未能转换 java.awt.FontRenderContext.getTransform
未能转换 java.awt.FontRenderContext.isAntiAliased
未能转换 java.awt.FontRenderContext.usesFractionalMetrics
未能转换 java.awt.Font.GlyphJustificationInfo
未能转换 java.awt.Font.GlyphMetrics
未能转换 java.awt.Font.GlyphVector
未能转换 java.awt.Font.GraphicAttribute
未能转换 java.awt.Font.GraphicAttribute.<AttributeName>
未能转换 java.awt.Font.GraphicAttribute.draw
未能转换 java.awt.Font.GraphicAttribute.getAdvance
未能转换 java.awt.Font.GraphicAttribute.getAlignment
未能转换 java.awt.Font.GraphicAttribute.getAscent
未能转换 java.awt.Font.GraphicAttribute.getBounds
未能转换 java.awt.Font.GraphicAttribute.getDescent
未能转换 java.awt.Font.GraphicAttribute.getJustificationInfo
未能转换 java.awt.Font.GraphicAttribute.GraphicAttribute
未能转换 java.awt.Font.ImageGraphicAttribute
未能转换 java.awt.Font.ImageGraphicAttribute.draw
未能转换 java.awt.Font.ImageGraphicAttribute.equals
未能转换 java.awt.Font.ImageGraphicAttribute.getAdvance
未能转换 java.awt.Font.ImageGraphicAttribute.getAscent
未能转换 java.awt.Font.ImageGraphicAttribute.getBounds
未能转换 java.awt.Font.ImageGraphicAttribute.getDescent
未能转换 java.awt.Font.ImageGraphicAttribute.hashCode
未能转换 java.awt.Font.ImageGraphicAttribute.ImageGraphicAttribute
未能转换 java.awt.Font.LineBreakMeasurer
未能转换 java.awt.Font.LineBreakMeasurer.deleteChar
未能转换 java.awt.Font.LineBreakMeasurer.getPosition
未能转换 java.awt.Font.LineBreakMeasurer.insertChar
未能转换 java.awt.Font.LineBreakMeasurer.LineBreakMeasurer
未能转换 java.awt.Font.LineBreakMeasurer.nextLayout
未能转换 java.awt.Font.LineBreakMeasurer.nextOffset
未能转换 java.awt.Font.LineBreakMeasurer.setPosition
未能转换 java.awt.Font.LineMetrics
未能转换 java.awt.Font.LineMetrics.getAscent

未能转换 java.awt.font.LineMetrics.getBaselineIndex
未能转换 java.awt.font.LineMetrics.getBaselineOffsets
未能转换 java.awt.font.LineMetrics.getDescent
未能转换 java.awt.font.LineMetrics.getHeight
未能转换 java.awt.font.LineMetrics.getLeading
未能转换 java.awt.font.LineMetrics.getNumChars
未能转换 java.awt.font.LineMetrics.getStrikethroughOffset
未能转换 java.awt.font.LineMetrics.getStrikethroughThickness
未能转换 java.awt.font.LineMetrics.getUnderlineOffset
未能转换 java.awt.font.LineMetrics.getUnderlineThickness
未能转换 java.awt.font.LineMetrics.LineMetrics
未能转换 java.awt.font.MultipleMaster
未能转换 java.awt.font.OpenType
未能转换 java.awt.font.OpenType.<TagType>
未能转换 java.awt.font.OpenType.getFontTable(int)
未能转换 java.awt.font.OpenType.getFontTable(int, int, int)
未能转换 java.awt.font.OpenType.getFontTable(String)
未能转换 java.awt.font.OpenType.getFontTable(String, int, int)
未能转换 java.awt.font.OpenType.getFontTableSize
未能转换 java.awt.font.OpenType.getVersion
未能转换 java.awt.font.ShapeGraphicAttribute
未能转换 java.awt.font.ShapeGraphicAttribute.draw
未能转换 java.awt.font.ShapeGraphicAttribute.equals
未能转换 java.awt.font.ShapeGraphicAttribute.FILL
未能转换 java.awt.font.ShapeGraphicAttribute.getAdvance
未能转换 java.awt.font.ShapeGraphicAttribute.getAscent
未能转换 java.awt.font.ShapeGraphicAttribute.getBounds
未能转换 java.awt.font.ShapeGraphicAttribute.getDescent
未能转换 java.awt.font.ShapeGraphicAttribute.hashCode
未能转换 java.awt.font.ShapeGraphicAttribute.ShapeGraphicAttribute
未能转换 java.awt.font.ShapeGraphicAttribute.STROKE
未能转换 java.awt.font.TextAttribute
未能转换 java.awt.font.TextHitInfo
未能转换 java.awt.font.TextHitInfo.afterOffset
未能转换 java.awt.font.TextHitInfo.beforeOffset
未能转换 java.awt.font.TextHitInfo.equals
未能转换 java.awt.font.TextHitInfo.getCharIndex
未能转换 java.awt.font.TextHitInfo.getInsertionIndex
未能转换 java.awt.font.TextHitInfo.getOffsetHit
未能转换 java.awt.font.TextHitInfo.getOtherHit

未能转换 java.awt.font.TextHitInfo.hashCode
未能转换 java.awt.font.TextHitInfo.isLeadingEdge
未能转换 java.awt.font.TextHitInfo.leading
未能转换 java.awt.font.TextHitInfo.toString
未能转换 java.awt.font.TextHitInfo.trailing
未能转换 java.awt.font.TextLayout
未能转换 java.awt.font.TextLayout.CaretPolicy
未能转换 java.awt.font.TextLayout.CaretPolicy.CaretPolicy
未能转换 java.awt.font.TextLayout.CaretPolicy.getStrongCaret
未能转换 java.awt.font.TextLayout.clone
未能转换 java.awt.font.TextLayout.DEFAULT_CARET_POLICY
未能转换 java.awt.font.TextLayout.draw
未能转换 java.awt.font.TextLayout.equals
未能转换 java.awt.font.TextLayout.getAdvance
未能转换 java.awt.font.TextLayout.getAscent
未能转换 java.awt.font.TextLayout.getBaseline
未能转换 java.awt.font.TextLayout.getBaselineOffsets
未能转换 java.awt.font.TextLayout.getBlackBoxBounds
未能转换 java.awt.font.TextLayout.getBounds
未能转换 java.awt.font.TextLayout.getCaretInfo
未能转换 java.awt.font.TextLayout.getCaretShape
未能转换 java.awt.font.TextLayout.getCaretShapes
未能转换 java.awt.font.TextLayout.getCharacterCount
未能转换 java.awt.font.TextLayout.getCharacterLevel
未能转换 java.awt.font.TextLayout.getDescent
未能转换 java.awt.font.TextLayout.getJustifiedLayout
未能转换 java.awt.font.TextLayout.getLeading
未能转换 java.awt.font.TextLayout.getLogicalHighlightShape
未能转换 java.awt.font.TextLayout.getLogicalRangesForVisualSelection
未能转换 java.awt.font.TextLayout.getNextLeftHit
未能转换 java.awt.font.TextLayout.getNextRightHit
未能转换 java.awt.font.TextLayout.getOutline
未能转换 java.awt.font.TextLayout.getVisibleAdvance
未能转换 java.awt.font.TextLayout.getVisualHighlightShape
未能转换 java.awt.font.TextLayout.getVisualOtherHit
未能转换 java.awt.font.TextLayout.handleJustify
未能转换 java.awt.font.TextLayout.hashCode
未能转换 java.awt.font.TextLayout.hitTestChar
未能转换 java.awt.font.TextLayout.isLeftToRight
未能转换 java.awt.font.TextLayout.isVertical

未能转换 java.awt.font.TextLayout.TextLayout

未能转换 java.awt.font.TextLayout.toString

未能转换 java.awt.font.TextMeasurer

未能转换 java.awt.font.TransformAttribute

Java.awt.geom 错误信息

未能转换 java.awt.geom.AffineTransform.<Type>
未能转换 java.awt.geom.AffineTransform.clone
未能转换 java.awt.geom.AffineTransform.concatenate
未能转换 java.awt.geom.AffineTransform.createTransformedShape
未能转换 java.awt.geom.AffineTransform.deltaTransform(double[], int, double[], int, int)
未能转换 java.awt.geom.AffineTransform.deltaTransform(Point2D, Point2D)
未能转换 java.awt.geom.AffineTransform.getType
未能转换 java.awt.geom.AffineTransform.inverseTransform(double[], int, double[], int, int)
未能转换 java.awt.geom.AffineTransform.inverseTransform(Point2D, Point2D)
未能转换 java.awt.geom.AffineTransform.preConcatenate
未能转换 java.awt.geom.AffineTransform.setTransform(AffineTransform)
未能转换 java.awt.geom.AffineTransform.setTransform(double, double, double, double, double, double)
未能转换 java.awt.geom.AffineTransform.transform
未能转换 java.awt.geom.Arc2D.Arc2D
未能转换 java.awt.geom.Arc2D.containsAngle
未能转换 java.awt.geom.Arc2D.Double.Double
未能转换 java.awt.geom.Arc2D.Double.extent
未能转换 java.awt.geom.Arc2D.Double.getAngleExtent
未能转换 java.awt.geom.Arc2D.Double.getAngleStart
未能转换 java.awt.geom.Arc2D.Double.getHeight
未能转换 java.awt.geom.Arc2D.Double.getWidth
未能转换 java.awt.geom.Arc2D.Double.getX
未能转换 java.awt.geom.Arc2D.Double.getY
未能转换 java.awt.geom.Arc2D.Double.height
未能转换 java.awt.geom.Arc2D.Double.makeBounds
未能转换 java.awt.geom.Arc2D.Double.setAngleExtent
未能转换 java.awt.geom.Arc2D.Double.setAngleStart
未能转换 java.awt.geom.Arc2D.Double.start
未能转换 java.awt.geom.Arc2D.Double.width
未能转换 java.awt.geom.Arc2D.Double.x
未能转换 java.awt.geom.Arc2D.Double.y
未能转换 java.awt.geom.Arc2D.Float.extent
未能转换 java.awt.geom.Arc2D.Float.Float
未能转换 java.awt.geom.Arc2D.Float.getAngleExtent
未能转换 java.awt.geom.Arc2D.Float.getAngleStart
未能转换 java.awt.geom.Arc2D.Float.getHeight
未能转换 java.awt.geom.Arc2D.Float.getWidth

未能转换 java.awt.geom.Arc2D.Float.getX
未能转换 java.awt.geom.Arc2D.Float.getY
未能转换 java.awt.geom.Arc2D.Float.height
未能转换 java.awt.geom.Arc2D.Float.makeBounds
未能转换 java.awt.geom.Arc2D.Float.setAngleExtent
未能转换 java.awt.geom.Arc2D.Float.setAngleStart
未能转换 java.awt.geom.Arc2D.Float.start
未能转换 java.awt.geom.Arc2D.Float.width
未能转换 java.awt.geom.Arc2D.Float.x
未能转换 java.awt.geom.Arc2D.Float.y
未能转换 java.awt.geom.Arc2D.getAngleExtent
未能转换 java.awt.geom.Arc2D.getAngleStart
未能转换 java.awt.geom.Arc2D.getArcType
未能转换 java.awt.geom.Arc2D.makeBounds
未能转换 java.awt.geom.Arc2D.setAngleExtent
未能转换 java.awt.geom.Arc2D.setAngles(double, double, double, double)
未能转换 java.awt.geom.Arc2D.setAngles(Point2D)
未能转换 java.awt.geom.Arc2D.setAngleStart(double)
未能转换 java.awt.geom.Arc2D.setAngleStart(Point2D)
未能转换 java.awt.geom.Arc2D.setArcByCenter
未能转换 java.awt.geom.Arc2D.setArcByTangent
未能转换 java.awt.geom.Arc2D.setArcType
未能转换 java.awt.geom.Arc2D setFrame
未能转换 java.awt.geom.Area.add
未能转换 java.awt.geom.Area.clone
未能转换 java.awt.geom.Area.contains
未能转换 java.awt.geom.Area.equals
未能转换 java.awt.geom.Area.exclusiveOr
未能转换 java.awt.geom.Area.getBounds
未能转换 java.awt.geom.Area.getBounds2D
未能转换 java.awt.geom.Area.getPathIterator
未能转换 java.awt.geom.Area.intersects
未能转换 java.awt.geom.Area.isEmpty
未能转换 java.awt.geom.Area.isPolygonal
未能转换 java.awt.geom.Area.isRectangular
未能转换 java.awt.geom.Area.isSingular
未能转换 java.awt.geom.Area.subtract
未能转换 java.awt.geom.CubicCurve2D.contains
未能转换 java.awt.geom.CubicCurve2D.CubicCurve2D
未能转换 java.awt.geom.CubicCurve2D.Double.Double

未能转换 java.awt.geom.CubicCurve2D.Float.Float
未能转换 java.awt.geom.CubicCurve2D.getBounds
未能转换 java.awt.geom.CubicCurve2D.getFlatness
未能转换 java.awt.geom.CubicCurve2D.getFlatnessSq
未能转换 java.awt.geom.CubicCurve2D.getPathIterator
未能转换 java.awt.geom.CubicCurve2D.solveCubic(double[])
未能转换 java.awt.geom.CubicCurve2D.solveCubic(double[], double[])
未能转换 java.awt.geom.CubicCurve2D.subdivide
未能转换 java.awt.geom.Dimension2D.clone
未能转换 java.awt.geom.Dimension2D.Dimension2D
未能转换 java.awt.geom.Ellipse2D.Double.Double
未能转换 java.awt.geom.Ellipse2D.Ellipse2D
未能转换 java.awt.geom.Ellipse2D.Float.Float
未能转换 java.awt.geom.FlatteningPathIterator
未能转换 java.awt.geom.GeneralPath.append
未能转换 java.awt.geom.GeneralPath.GeneralPath
未能转换 java.awt.geom.GeneralPath.GeneralPath(int)
未能转换 java.awt.geom.GeneralPath.GeneralPath(int, int)
未能转换 java.awt.geom.GeneralPath.GeneralPath(Shape)
未能转换 java.awt.geom.GeneralPath.getBounds
未能转换 java.awt.geom.GeneralPath.getBounds2D
未能转换 java.awt.geom.GeneralPath.getCurrentPoint
未能转换 java.awt.geom.GeneralPath.getPathIterator
未能转换 java.awt.geom.GeneralPath.moveTo
未能转换 java.awt.geom.GeneralPath.quadTo
未能转换 java.awt.geom.Line2D.Double.Double(double, double, double, double)
未能转换 java.awt.geom.Line2D.Double.Double(Point2D, Point2D)
未能转换 java.awt.geom.Line2D.Float.Float(float, float, float, float)
未能转换 java.awt.geom.Line2D.Float.Float(Point2D, Point2D)
未能转换 java.awt.geom.Line2D.getBounds
未能转换 java.awt.geom.Line2D.getPathIterator
未能转换 java.awt.geom.Line2D.intersectsLine
未能转换 java.awt.geom.Line2D.Line2D
未能转换 java.awt.geom.Line2D.linesIntersect
未能转换 java.awt.geom.Line2D.ptLineDist
未能转换 java.awt.geom.Line2D.ptLineDistSq
未能转换 java.awt.geom.Line2D.ptSegDist
未能转换 java.awt.geom.Line2D.ptSegDistSq
未能转换 java.awt.geom.Line2D.relativeCCW
未能转换 java.awt.geom.PathIterator

未能转换 java.awt.geom.PathIterator.<SegmentType>
未能转换 java.awt.geom.PathIterator.currentSegment
未能转换 java.awt.geom.PathIterator.getWindingRule
未能转换 java.awt.geom.PathIterator.isDone
未能转换 java.awt.geom.PathIterator.next
未能转换 java.awt.geom.Point2D.Point2D
未能转换 java.awt.geom.QuadCurve2D
未能转换 java.awt.geom.QuadCurve2D.Double
未能转换 java.awt.geom.QuadCurve2D.Float
未能转换 java.awt.geom.Rectangle2D.<Position>
未能转换 java.awt.geom.Rectangle2D.add
未能转换 java.awt.geom.Rectangle2D.createIntersection
未能转换 java.awt.geom.Rectangle2D.Double.createIntersection
未能转换 java.awt.geom.Rectangle2D.Double.outcode
未能转换 java.awt.geom.Rectangle2D.Float.createIntersection
未能转换 java.awt.geom.Rectangle2D.Float.outcode
未能转换 java.awt.geom.Rectangle2D.getPathIterator
未能转换 java.awt.geom.Rectangle2D.intersect
未能转换 java.awt.geom.Rectangle2D.intersectsLine
未能转换 java.awt.geom.Rectangle2D.outcode
未能转换 java.awt.geom.Rectangle2D.Rectangle2D
未能转换 java.awt.geom.RectangularShape.clone
未能转换 java.awt.geom.RectangularShape.getBounds
未能转换 java.awt.geom.RectangularShape.getPathIterator
未能转换 java.awt.geom.RectangularShape setFrame(double, double, double, double)
未能转换 java.awt.geom.RectangularShape setFrame(Point2D, Dimension2D)
未能转换 java.awt.geom.RectangularShape setFrame(Rectangle2D)
未能转换 java.awt.geom.RectangularShape setFrameFromCenter
未能转换 java.awt.geom.RectangularShape setFrameFromDiagonal(double, double, double, double)
未能转换 java.awt.geom.RectangularShape setFrameFromDiagonal(Point2D, Point2D)
未能转换 java.awt.geom.RoundRectangle2D.Double
未能转换 java.awt.geom.RoundRectangle2D.Double.archeight
未能转换 java.awt.geom.RoundRectangle2D.Double.arcwidth
未能转换 java.awt.geom.RoundRectangle2D.Double.getArcHeight
未能转换 java.awt.geom.RoundRectangle2D.Double.getArcWidth
未能转换 java.awt.geom.RoundRectangle2D.Double.height
未能转换 java.awt.geom.RoundRectangle2D.Double.setRoundRect
未能转换 java.awt.geom.RoundRectangle2D.Double.width
未能转换 java.awt.geom.RoundRectangle2D.Double.x
未能转换 java.awt.geom.RoundRectangle2D.Double.y

未能转换 java.awt.geom.RoundRectangle2D.Float
未能转换 java.awt.geom.RoundRectangle2D.Float.archeight
未能转换 java.awt.geom.RoundRectangle2D.Float.arcwidth
未能转换 java.awt.geom.RoundRectangle2D.Float.getArcHeight
未能转换 java.awt.geom.RoundRectangle2D.Float.getArcWidth
未能转换 java.awt.geom.RoundRectangle2D.Float.height
未能转换 java.awt.geom.RoundRectangle2D.Float.setRoundRect
未能转换 java.awt.geom.RoundRectangle2D.Float.width
未能转换 java.awt.geom.RoundRectangle2D.Float.x
未能转换 java.awt.geom.RoundRectangle2D.Float.y
未能转换 java.awt.geom.RoundRectangle2D.getArcHeight
未能转换 java.awt.geom.RoundRectangle2D.getArcWidth
未能转换 java.awt.geom.RoundRectangle2D.RoundRectangle2D
未能转换 java.awt.geom.RoundRectangle2D setFrame
未能转换 java.awt.geom.RoundRectangle2D.setRoundRect

Java.awt.im 错误信息

未能转换 java.awt.im.InputContext
未能转换 java.awt.im.InputMethodHighlight
未能转换 java.awt.im.InputMethodRequests
未能转换 java.awt.im.InputSubset
未能转换 java.awt.im.spi.InputMethod
未能转换 java.awt.im.spi.InputMethodContext
未能转换 java.awt.im.spi.InputMethodDescriptor

Java.awt.image 错误信息

未能转换 java.awt.image.<ClassName>.*Consumer
未能转换 java.awt.image.<ClassName>.*TopDownLeftRight*
未能转换 java.awt.image.<ClassName>.imageComplete
未能转换 java.awt.image.<ClassName>.setHints
未能转换 java.awt.image.<ClassName>.setPixels
未能转换 java.awt.image.<ClassName>.setProperties
未能转换 java.awt.image.AffineTransformOp
未能转换 java.awt.image.BandCombineOp
未能转换 java.awt.image.BandedSampleModel
未能转换 java.awt.image.BandedSampleModel.BandedSampleModel
未能转换 java.awt.image.BandedSampleModel.createCompatibleSampleModel
未能转换 java.awt.image.BandedSampleModel.createSubsetSampleModel
未能转换 java.awt.image.BandedSampleModel.getDataElements
未能转换 java.awt.image.BandedSampleModel.getPixel
未能转换 java.awt.image.BandedSampleModel.getPixels
未能转换 java.awt.image.BandedSampleModel.getSample
未能转换 java.awt.image.BandedSampleModel.getSampleDouble
未能转换 java.awt.image.BandedSampleModel.getSampleFloat
未能转换 java.awt.image.BandedSampleModel.getSamples
未能转换 java.awt.image.BandedSampleModel.setDataElements
未能转换 java.awt.image.BandedSampleModel.setSample
未能转换 java.awt.image.BandedSampleModel.setSamples
未能转换 java.awt.image.BufferedImage.addTileObserver
未能转换 java.awt.image.BufferedImage.BufferedImage(ColorModel, Raster, Hashtable)
未能转换 java.awt.image.BufferedImage.BufferedImage(int, int, int)
未能转换 java.awt.image.BufferedImage.BufferedImage(int, int, int, ColorModel)
未能转换 java.awt.image.BufferedImage.coerceData
未能转换 java.awt.image.BufferedImage.copyData
未能转换 java.awt.image.BufferedImage.getAlphaRaster
未能转换 java.awt.image.BufferedImage.getColorModel
未能转换 java.awt.image.BufferedImage.getHeight
未能转换 java.awt.image.BufferedImage.getMinTileX
未能转换 java.awt.image.BufferedImage.getMinTileY
未能转换 java.awt.image.BufferedImage.getNumXTiles
未能转换 java.awt.image.BufferedImage.getNumYTiles
未能转换 java.awt.image.BufferedImage.getProperty
未能转换 java.awt.image.BufferedImage.getPropertyNames

未能转换 java.awt.image.BufferedImage.getRGB
未能转换 java.awt.image.BufferedImage.getSources
未能转换 java.awt.image.BufferedImage.getTile
未能转换 java.awt.image.BufferedImage.getTileGridXOffset
未能转换 java.awt.image.BufferedImage.getTileGridYOffset
未能转换 java.awt.image.BufferedImage.getTileHeight
未能转换 java.awt.image.BufferedImage.getTileWidth
未能转换 java.awt.image.BufferedImage.getWidth
未能转换 java.awt.image.BufferedImage.getWritableTile
未能转换 java.awt.image.BufferedImage.getWritableTileIndices
未能转换 java.awt.image.BufferedImage.hasTileWriters
未能转换 java.awt.image.BufferedImage.isAlphaPremultiplied
未能转换 java.awt.image.BufferedImage.isTileWritable
未能转换 java.awt.image.BufferedImage.releaseWritableTile
未能转换 java.awt.image.BufferedImage.removeTileObserver
未能转换 java.awt.image.BufferedImage.setRGB
未能转换 java.awt.image.BufferedImage.TYPE_BYTE_BINARY
未能转换 java.awt.image.BufferedImage.TYPE_BYTE_GRAY
未能转换 java.awt.image BufferedImageFilter
未能转换 java.awt.image.BufferedImageOp
未能转换 java.awt.image.ByteLookupTable
未能转换 java.awt.image.ColorConvertOp
未能转换 java.awt.image.ConvolveOp
未能转换 java.awt.image.ColorModel.coerceData
未能转换 java.awt.image.ColorModel.ColorModel
未能转换 java.awt.image.ColorModel.createCompatibleSampleModel
未能转换 java.awt.image.ColorModel.createCompatibleWritableRaster
未能转换 java.awt.image.ColorModel.finalize
未能转换 java.awt.image.ColorModel.getAlpha
未能转换 java.awt.image.ColorModel.getAlphaRaster
未能转换 java.awt.image.ColorModel.getBlue
未能转换 java.awt.image.ColorModel.getColorSpace
未能转换 java.awt.image.ColorModel.getComponents(int, int[], int)
未能转换 java.awt.image.ColorModel.getComponents(object, int[], int)
未能转换 java.awt.image.ColorModel.getComponentSize
未能转换 java.awt.image.ColorModel.getComponentSize(int)
未能转换 java.awt.image.ColorModel.getDataElement
未能转换 java.awt.image.ColorModel.getDataElements(int, Object)
未能转换 java.awt.image.ColorModel.getDataElements(int[], int, Object)
未能转换 java.awt.image.ColorModel.getGreen

未能转换 java.awt.image.ColorModel.getNormalizedComponents
未能转换 java.awt.image.ColorModel.getNumColorComponents
未能转换 java.awt.image.ColorModel.getNumComponents
未能转换 java.awt.image.ColorModel.getRed
未能转换 java.awt.image.ColorModel.getRGB
未能转换 java.awt.image.ColorModel.getTransferType
未能转换 java.awt.image.ColorModel.getTransparency
未能转换 java.awt.image.ColorModel.getUnnormalizedComponents
未能转换 java.awt.image.ComponentColorModel.coerceData
未能转换 java.awt.image.ComponentColorModel.ComponentColorModel
未能转换 java.awt.image.ComponentColorModel.createCompatibleSampleModel
未能转换 java.awt.image.ComponentColorModel.createCompatibleWritableRaster
未能转换 java.awt.image.ComponentColorModel.getAlpha
未能转换 java.awt.image.ComponentColorModel.getAlphaRaster
未能转换 java.awt.image.ComponentColorModel.getBlue
未能转换 java.awt.image.ComponentColorModel.getComponents
未能转换 java.awt.image.ComponentColorModel.getDataElement
未能转换 java.awt.image.ComponentColorModel.getDataElements(int[], int)
未能转换 java.awt.image.ComponentColorModel.getDataElements(int, Object)
未能转换 java.awt.image.ComponentColorModel.getGreen
未能转换 java.awt.image.ComponentColorModel.getRed
未能转换 java.awt.image.ComponentColorModel.getRGB
未能转换 java.awt.image.ColorModel.isAlphaPremultiplied
未能转换 java.awt.image.ColorModel.isCompatibleRaster
未能转换 java.awt.image.ColorModel.isCompatibleSampleModel
未能转换 java.awt.image.ColorModel.transferType
未能转换 java.awt.image.ComponentColorModel.isCompatibleRaster
未能转换 java.awt.image.ComponentColorModel.isCompatibleSampleModel
未能转换 java.awt.image.ComponentSampleModel
未能转换 java.awt.image.ComponentSampleModel.bandOffsets
未能转换 java.awt.image.ComponentSampleModel.bankIndices
未能转换 java.awt.image.ComponentSampleModel.ComponentSampleModel
未能转换 java.awt.image.ComponentSampleModel.createCompatibleSampleModel
未能转换 java.awt.image.ComponentSampleModel.createSubsetSampleModel
未能转换 java.awt.image.ComponentSampleModel.getBandOffsets
未能转换 java.awt.image.ComponentSampleModel.getBankIndices
未能转换 java.awt.image.ComponentSampleModel.getDataElements
未能转换 java.awt.image.ComponentSampleModel.getNumDataElements
未能转换 java.awt.image.ComponentSampleModel.getOffset(int, int)
未能转换 java.awt.image.ComponentSampleModel.getOffset(int, int, int)

未能转换 java.awt.image.ComponentSampleModel.getPixel
未能转换 java.awt.image.ComponentSampleModel.getPixels
未能转换 java.awt.image.ComponentSampleModel.getSample
未能转换 java.awt.image.ComponentSampleModel.getSampleDouble
未能转换 java.awt.image.ComponentSampleModel.getSampleFloat
未能转换 java.awt.image.ComponentSampleModel.getSamples
未能转换 java.awt.image.ComponentSampleModel.getSampleSize
未能转换 java.awt.image.ComponentSampleModel.numBands
未能转换 java.awt.image.ComponentSampleModel.numBanks
未能转换 java.awt.image.ComponentSampleModel.pixelStride
未能转换 java.awt.image.ComponentSampleModel.scanlineStride
未能转换 java.awt.image.ComponentSampleModel.setDataElements
未能转换 java.awt.image.ComponentSampleModel.setSample
未能转换 java.awt.image.ComponentSampleModel.setSamples
未能转换 java.awt.image.DataBuffer.<DataType>
未能转换 java.awt.image.DataBuffer.DataBuffer(int, int, int, int)
未能转换 java.awt.image.DataBuffer.DataBuffer(int, int, int, int[])
未能转换 java.awt.image.DataBuffer.dataType
未能转换 java.awt.image.DataBuffer.getDataType
未能转换 java.awt.image.DataBuffer.getDataTypeSize
未能转换 java.awt.image.DataBuffer.getElem
未能转换 java.awt.image.DataBuffer.getElemDouble
未能转换 java.awt.image.DataBuffer.getElemFloat
未能转换 java.awt.image.DataBuffer.getOffset
未能转换 java.awt.image.DataBuffer.getOffsets
未能转换 java.awt.image.DataBuffer.offset
未能转换 java.awt.image.DataBuffer.offsets
未能转换 java.awt.image.DataBuffer.setElem
未能转换 java.awt.image.DataBuffer.setElemDouble
未能转换 java.awt.image.DataBuffer.setElemFloat
未能转换 java.awt.image.DataBufferByte.DataBufferByte
未能转换 java.awt.image.DataBufferByte.getBankData
未能转换 java.awt.image.DataBufferInt.DataBufferInt
未能转换 java.awt.image.DataBufferInt.getBankData
未能转换 java.awt.image.DataBufferShort.DataBufferShort
未能转换 java.awt.image.DataBufferShort.getBankData
未能转换 java.awt.image.DataBufferUShort.DataBufferUShort
未能转换 java.awt.image.DataBufferUShort.getBankData
未能转换 java.awt.image.DirectColorModel.coerceData
未能转换 java.awt.image.DirectColorModel.createCompatibleWritableRaster

未能转换 java.awt.image.DirectColorModel.DirectColorModel(ColorSpace, int, int, int, int, int, boolean, int)
未能转换 java.awt.image.DirectColorModel.DirectColorModel(int, int, int, int)
未能转换 java.awt.image.DirectColorModel.DirectColorModel(int, int, int, int, int, int)
未能转换 java.awt.image.DirectColorModel.getAlpha
未能转换 java.awt.image.DirectColorModel.getBlue
未能转换 java.awt.image.DirectColorModel.getComponents
未能转换 java.awt.image.DirectColorModel.getDataElement
未能转换 java.awt.image.DirectColorModel.getDataElements(int, Object)
未能转换 java.awt.image.DirectColorModel.getDataElements(int[], int)
未能转换 java.awt.image.DirectColorModel.getGreen
未能转换 java.awt.image.DirectColorModel.getRed
未能转换 java.awt.image.DirectColorModel.getRGB
未能转换 java.awt.image.DirectColorModel.isCompatibleRaster
未能转换 java.awt.image.FilteredImageSource.startProduction
未能转换 java.awt.image.ImageConsumer
未能转换 java.awt.image.ImageFilter.setColorModel
未能转换 java.awt.image.ImageObserver
未能转换 java.awt.image.ImageProducer.startProduction
未能转换 java.awt.image.ImagingOpException
未能转换 java.awt.image.IndexColorModel.convertToIntDiscrete
未能转换 java.awt.image.IndexColorModel.createCompatibleSampleModel
未能转换 java.awt.image.IndexColorModel.createCompatibleWritableRaster
未能转换 java.awt.image.IndexColorModel.finalize
未能转换 java.awt.image.IndexColorModel.getComponents
未能转换 java.awt.image.IndexColorModel.getComponentSize
未能转换 java.awt.image.IndexColorModel.getDataElement
未能转换 java.awt.image.IndexColorModel.getDataElements
未能转换 java.awt.image.IndexColorModel.getRBGs
未能转换 java.awt.image.IndexColorModel.getTransparency
未能转换 java.awt.image.IndexColorModel.getValidPixels
未能转换 java.awt.image.IndexColorModel.IndexColorModel(int, int, int[], int, boolean,int,int)
未能转换 java.awt.image.IndexColorModel.IndexColorModel(int, int, int[], int, int, BigInteger)
未能转换 java.awt.image.IndexColorModel.isCompatibleRaster
未能转换 java.awt.image.IndexColorModel.isCompatibleSampleModel
未能转换 java.awt.image.Kernel
未能转换 java.awt.image.LookupOp
未能转换 java.awt.image.LookupTable
未能转换 java.awt.image.MemoryImageSource
未能转换 java.awt.image.MultiPixelPackedSampleModel
未能转换 java.awt.image.MultiPixelPackedSampleModel.createCompatibleSampleModel

未能转换 java.awt.image.MultiPixelPackedSampleModel.createSubsetSampleModel
未能转换 java.awt.image.MultiPixelPackedSampleModel.getBitOffset
未能转换 java.awt.image.MultiPixelPackedSampleModel.getDataBitOffset
未能转换 java.awt.image.MultiPixelPackedSampleModel.getDataElements
未能转换 java.awt.image.MultiPixelPackedSampleModel.getNumDataElements
未能转换 java.awt.image.MultiPixelPackedSampleModel.getOffset
未能转换 java.awt.image.MultiPixelPackedSampleModel.getPixel
未能转换 java.awt.image.MultiPixelPackedSampleModel.getSample
未能转换 java.awt.image.MultiPixelPackedSampleModel.getSampleSize
未能转换 java.awt.image.MultiPixelPackedSampleModel.getSampleSize(int)
未能转换 java.awt.image.MultiPixelPackedSampleModel.getTransferType
未能转换 java.awt.image.MultiPixelPackedSampleModel.MultiPixelPackedSampleModel(int, int, int, int)
未能转换 java.awt.image.MultiPixelPackedSampleModel.MultiPixelPackedSampleModel(int, int, int, int, int, int)
未能转换 java.awt.image.MultiPixelPackedSampleModel.setDataElements
未能转换 java.awt.image.MultiPixelPackedSampleModel.setSample
未能转换 java.awt.image.PackedColorModel.createCompatibleSampleModel
未能转换 java.awt.image.PackedColorModel.getAlphaRaster
未能转换 java.awt.image.PackedColorModel.getMask
未能转换 java.awt.image.PackedColorModel.getMasks
未能转换 java.awt.image.PackedColorModel.isCompatibleSampleModel
未能转换 java.awt.image.PackedColorModel.PackedColorModel
未能转换 java.awt.image.PixelGrabber.abortGrabbing
未能转换 java.awt.image.PixelGrabber.getStatus
未能转换 java.awt.image.PixelGrabber.setColorModel
未能转换 java.awt.image.PixelGrabber.startGrabbing
未能转换 java.awt.image.PixelGrabber.status
未能转换 java.awt.image.PixelInterleavedSampleModel
未能转换 java.awt.image.PixelInterleavedSampleModel.createCompatibleSampleModel
未能转换 java.awt.image.PixelInterleavedSampleModel.createSubsetSampleModel
未能转换 java.awt.image.PixelInterleavedSampleModel.PixelInterleavedSampleModel
未能转换 java.awt.image.Raster
未能转换 java.awt.image.Raster.createChild
未能转换 java.awt.image.Raster.createCompatibleWritableRaster
未能转换 java.awt.image.Raster.createRaster
未能转换 java.awt.image.Raster.createTranslatedChild
未能转换 java.awt.image.Raster.createWritableRaster
未能转换 java.awt.image.Raster.getDataBuffer
未能转换 java.awt.image.Raster.getDataElements(int, int, int, int, object)
未能转换 java.awt.image.Raster.getDataElements(int, int, object)
未能转换 java.awt.image.Raster.getNumBands

未能转换 java.awt.image.Raster.getNumDataElements
未能转换 java.awt.image.Raster.getParent
未能转换 java.awt.image.Raster.getPixel(int, int, double[])
未能转换 java.awt.image.Raster.getPixel(int, int, float[])
未能转换 java.awt.image.Raster.getPixels(int, int, int, int, double[])
未能转换 java.awt.image.Raster.getPixels(int, int, int, int, float[])
未能转换 java.awt.image.Raster.getPixels(int, int, int, int, int[])
未能转换 java.awt.image.Raster.getPixel(int, int, int[])
未能转换 java.awt.image.Raster.getSample
未能转换 java.awt.image.Raster.getSampleDouble
未能转换 java.awt.image.Raster.getSampleFloat
未能转换 java.awt.image.Raster.getSampleModelTranslateX
未能转换 java.awt.image.Raster.getSampleModelTranslateY
未能转换 java.awt.image.Raster.getSamples
未能转换 java.awt.image.Raster.getSamples(int, int, int, int, int, double[])
未能转换 java.awt.image.Raster.getSamples(int, int, int, int, int, float[])
未能转换 java.awt.image.Raster.getTransferType
未能转换 java.awt.image.Raster.numBands
未能转换 java.awt.image.Raster.numDataElements
未能转换 java.awt.image.Raster.parent
未能转换 java.awt.image.Raster(Raster, SampleModel, DataBuffer, Point)
未能转换 java.awt.image.Raster(Raster, SampleModel, DataBuffer, Rectangle, Point, Raster)
未能转换 java.awt.image.Raster(SampleModel, Point)
未能转换 java.awt.image.Raster.sampleModelTranslateX
未能转换 java.awt.image.Raster.sampleModelTranslateY
未能转换 java.awt.image.RasterOp
未能转换 java.awt.image.renderable.ContextualRenderedImageFactory
未能转换 java.awt.image.renderable.RenderableImage.createDefaultRendering
未能转换 java.awt.image.renderable.RenderableImage.createRendering
未能转换 java.awt.image.renderable.RenderableImage.createScaledRendering
未能转换 java.awt.image.renderable.RenderableImage.getMinX
未能转换 java.awt.image.renderable.RenderableImage.getMinY
未能转换 java.awt.image.renderable.RenderableImage.getProperty
未能转换 java.awt.image.renderable.RenderableImage.getPropertyNames
未能转换 java.awt.image.renderable.RenderableImage.getSources
未能转换 java.awt.image.renderable.RenderableImage.HINTS_OBSERVED
未能转换 java.awt.image.renderable.RenderableImage.isDynamic
未能转换 java.awt.image.renderable.RenderableImageOp
未能转换 java.awt.image.renderable.RenderableImageProducer.addConsumer
未能转换 java.awt.image.renderable.RenderableImageProducer.isConsumer

未能转换 java.awt.image.renderable.RenderableImageProducer.removeConsumer
未能转换 java.awt.image.renderable.RenderableImageProducer.RenderableImageProducer
未能转换 java.awt.image.renderable.RenderableImageProducer.requestTopDownLeftRightResend
未能转换 java.awt.image.renderable.RenderableImageProducer.run
未能转换 java.awt.image.renderable.RenderableImageProducer.setRenderContext
未能转换 java.awt.image.renderable.RenderableImageProducer.startProduction
未能转换 java.awt.image.renderable.RenderContext
未能转换 java.awt.image.renderable.RenderedImageFactory
未能转换 java.awt.image.RenderedImage.copyData
未能转换 java.awt.image.RenderedImage.getColorModel
未能转换 java.awt.image.RenderedImage.getMinTileX
未能转换 java.awt.image.RenderedImage.getMinTileY
未能转换 java.awt.image.RenderedImage.getNumXTiles
未能转换 java.awt.image.RenderedImage.getNumYTiles
未能转换 java.awt.image.RenderedImage.getProperty
未能转换 java.awt.image.RenderedImage.getPropertyNames
未能转换 java.awt.image.RenderedImage.getSources
未能转换 java.awt.image.RenderedImage.getTile
未能转换 java.awt.image.RenderedImage.getTileGridXOffset
未能转换 java.awt.image.RenderedImage.getTileGridYOffset
未能转换 java.awt.image.RenderedImage.getTileHeight
未能转换 java.awt.image.RenderedImage.getTileWidth
未能转换 java.awt.image.ReplicateScaleFilter.outpixbuf
未能转换 java.awt.image.ReplicateScaleFilter.src*
未能转换 java.awt.image.RescaleOp
未能转换 java.awt.image.RGBImageFilter
未能转换 java.awt.image.SampleModel
未能转换 java.awt.image.SampleModel.createCompatibleSampleModel
未能转换 java.awt.image.SampleModel.createSubsetSampleModel
未能转换 java.awt.image.SampleModel.dataType
未能转换 java.awt.image.SampleModel.getDataElements(int, int, int, int, Object, DataBuffer)
未能转换 java.awt.image.SampleModel.getDataElements(int, int, Object, DataBuffer)
未能转换 java.awt.image.SampleModel.getDataType
未能转换 java.awt.image.SampleModel.getNumBands
未能转换 java.awt.image.SampleModel.getNumDataElements
未能转换 java.awt.image.SampleModel.getPixel(int, int, double[], DataBuffer)
未能转换 java.awt.image.SampleModel.getPixel(int, int, float[], DataBuffer)
未能转换 java.awt.image.SampleModel.getPixel(int, int, int[], DataBuffer)
未能转换 java.awt.image.SampleModel.getPixels(int, int, int, int, double[], DataBuffer)
未能转换 java.awt.image.SampleModel.getPixels(int, int, int, int, float[], DataBuffer)

未能转换 java.awt.image.SampleModel.getPixels(int, int, int, int, int[], DataBuffer)
未能转换 java.awt.image.SampleModel.getSample
未能转换 java.awt.image.SampleModel.getSampleDouble
未能转换 java.awt.image.SampleModel.getSampleFloat
未能转换 java.awt.image.SampleModel.getSamples(int, int, int, int, double[], DataBuffer)
未能转换 java.awt.image.SampleModel.getSamples(int, int, int, int, float[], DataBuffer)
未能转换 java.awt.image.SampleModel.getSamples(int, int, int, int, int, int[], DataBuffer)
未能转换 java.awt.image.SampleModel.getSampleSize
未能转换 java.awt.image.SampleModel.getSampleSize(int)
未能转换 java.awt.image.SampleModel.getTransferType
未能转换 java.awt.image.SampleModel.numBands
未能转换 java.awt.image.SampleModel.SampleModel
未能转换 java.awt.image.SampleModel.setDataElements(int, int, int, int, Object, DataBuffer)
未能转换 java.awt.image.SampleModel.setDataElements(int, int, Object, DataBuffer)
未能转换 java.awt.image.SampleModel.setPixel
未能转换 java.awt.image.SampleModel.setSample
未能转换 java.awt.image.SampleModel.setSamples
未能转换 java.awt.image.ShortLookupTable
未能转换 java.awt.image.SinglePixelPackedSampleModel
未能转换 java.awt.image.SinglePixelPackedSampleModel.createCompatibleSampleModel
未能转换 java.awt.image.SinglePixelPackedSampleModel.createSubsetSampleModel
未能转换 java.awt.image.SinglePixelPackedSampleModel.getBitMasks
未能转换 java.awt.image.SinglePixelPackedSampleModel.getBitOffsets
未能转换 java.awt.image.SinglePixelPackedSampleModel.getDataElements
未能转换 java.awt.image.SinglePixelPackedSampleModel.getNumDataElements
未能转换 java.awt.image.SinglePixelPackedSampleModel.getOffset
未能转换 java.awt.image.SinglePixelPackedSampleModel.getPixel
未能转换 java.awt.image.SinglePixelPackedSampleModel.getPixels
未能转换 java.awt.image.SinglePixelPackedSampleModel.getSample
未能转换 java.awt.image.SinglePixelPackedSampleModel.getSamples
未能转换 java.awt.image.SinglePixelPackedSampleModel.getSampleSize
未能转换 java.awt.image.SinglePixelPackedSampleModel.getSampleSize(int)
未能转换 java.awt.image.SinglePixelPackedSampleModel.setDataElements
未能转换 java.awt.image.SinglePixelPackedSampleModel.setSample
未能转换 java.awt.image.SinglePixelPackedSampleModel.setSamples
未能转换 java.awt.image.SinglePixelPackedSampleModel.SinglePixelPackedSampleModel(int, int, int, int, int[])
未能转换 java.awt.image.SinglePixelPackedSampleModel.SinglePixelPackedSampleModel(int, int, int, int, int[])
未能转换 java.awt.image.TileObserver
未能转换 java.awt.Image.UndefinedProperty
未能转换 java.awt.image.WritableRaster

未能转换 java.awt.image.WritableRaster.createWritableChild
未能转换 java.awt.image.WritableRaster.createWritableTranslatedChild
未能转换 java.awt.image.WritableRaster.getParent
未能转换 java.awt.image.WritableRaster.setDataElements(int, int, int, int, Object)
未能转换 java.awt.image.WritableRaster.setDataElements(int, int, Object)
未能转换 java.awt.image.WritableRaster.setDataElements(int, int, Raster)
未能转换 java.awt.image.WritableRaster.setPixel(int, int, int, double)
未能转换 java.awt.image.WritableRaster.setPixel(int, int, int, float)
未能转换 java.awt.image.WritableRaster.setPixel(int, int, int, int)
未能转换 java.awt.image.WritableRaster.setRect
未能转换 java.awt.image.WritableRaster.setSample(int, int, int, double)
未能转换 java.awt.image.WritableRaster.setSample(int, int, int, float)
未能转换 java.awt.image.WritableRaster.setSample(int, int, int, int)
未能转换 java.awt.image.WritableRaster.setSamples(int, int, int, int, int, double[])
未能转换 java.awt.image.WritableRaster.setSamples(int, int, int, int, int, float[])
未能转换 java.awt.image.WritableRaster.setSamples(int, int, int, int, int, int[])
未能转换 java.awt.image.WritableRaster.WritableRaster(Rectangle, Point, WritableRaster)
未能转换 java.awt.image.WritableRaster.WritableRaster(SampleModel, DataBuffer, Point)
未能转换 java.awt.image.WritableRaster.WritableRaster(SampleModel, Point)
未能转换 java.awt.image.WritableRenderedImage

Java.awt.peer 错误信息

未能转换 java.awt.peer.CanvasPeer
未能转换 java.awt.peer.ComponentPeer.checkImage
未能转换 java.awt.peer.ComponentPeer.coalescePaintEvent
未能转换 java.awt.peer.ComponentPeer.createImage (ImageProducer)
未能转换 java.awt.peer.ComponentPeer.createImage(int, int)
未能转换 java.awt.peer.ComponentPeer.getColorModel
未能转换 java.awt.peer.ComponentPeer.getGraphicsConfiguration
未能转换 java.awt.peer.ComponentPeer.getMinimumSize
未能转换 java.awt.peer.ComponentPeer.getPreferredSize
未能转换 java.awt.peer.ComponentPeer.getToolkit
未能转换 java.awt.peer.ComponentPeer.handleEvent
未能转换 java.awt.peer.ComponentPeer.minimumSize
未能转换 java.awt.peer.ComponentPeer.paint
未能转换 java.awt.peer.ComponentPeer.preferredSize
未能转换 java.awt.peer.ComponentPeer.prepareImage
未能转换 java.awt.peer.ComponentPeer.print
未能转换 java.awt.peer.ComponentPeer.setBounds
未能转换 java.awt.peer.ComponentPeer.setVisible
未能转换 java.awt.peer.ContainerPeer.beginValidate
未能转换 java.awt.peer.ContainerPeer.endValidate
未能转换 java.awt.peer.FileDialogPeer.setFilenameFilter
未能转换 java.awt.peer.FramePeer.setIconImage
未能转换 java.awt.peer.LightweightPeer
未能转换 java.awt.peer.ListPeer.deleteItems
未能转换 java.awt.peer.ListPeer.makeVisible
未能转换 java.awt.peer.MenuBarPeer.addHelpMenu
未能转换 java.awt.peer.PopupMenuPeer.show
未能转换 java.awt.peer.RobotPeer
未能转换 java.awt.peer.ScrollbarPeer.setValues
未能转换 java.awt.peer.ScrollPanePeer.setUnitIncrement
未能转换 java.awt.peer.ScrollPanePeer.setValue
未能转换 java.awt.peer.TextComponentPeer.filterEvents
未能转换 java.awt.peer.TextComponentPeer.getCharacterBounds
未能转换 java.awt.peer.TextComponentPeer.getIndexAtPoint
未能转换 java.awt.peer.TextFieldPeer.getPreferredSize
未能转换 java.awt.peer.TextFieldPeer.preferredSize
未能转换 java.awt.peer.WindowPeer.handleFocusTraversalEvent

Java.awt.print 错误信息

未能转换 java.awt.print.Book
未能转换 java.awt.print.Pageable
未能转换 java.awt.print.PageFormat.<Format>
未能转换 java.awt.print.PageFormat.getMatrix
未能转换 java.awt.print.PageFormat.setOrientation
未能转换 java.awt.print.Paper.clone
未能转换 java.awt.print.Paper.getImageableHeight
未能转换 java.awt.print.Paper.getImageableWidth
未能转换 java.awt.print.Paper.getImageableX
未能转换 java.awt.print.Paper.getImageableY
未能转换 java.awt.print.Paper.Paper
未能转换 java.awt.print.Paper.setImageableArea
未能转换 java.awt.print.Paper.setSize
未能转换 java.awt.print.Printable
未能转换 java.awt.print.PrinterGraphics
未能转换 java.awt.print.PrinterJob.cancel
未能转换 java.awt.print.PrinterJob.defaultPage
未能转换 java.awt.print.PrinterJob.isCancelled
未能转换 java.awt.print.PrinterJob.setPageable
未能转换 java.awt.print.PrinterJob.setPrintable(Printable)
未能转换 java.awt.print.PrinterJob.setPrintable(Printable, PageFormat)
未能转换 java.awt.print.PrinterJob.validatePage

Java.beans 错误信息

未能转换 java.beans.AppletInitializer
未能转换 java.beans.BeanDescriptor
未能转换 java.beans.BeanDescriptor.BeanDescriptor
未能转换 java.beans.BeanDescriptor.getCustomizerClass
未能转换 java.beans.BeanInfo
未能转换 java.beans.BeanInfo.ICON_COLOR_16x16
未能转换 java.beans.BeanInfo.ICON_COLOR_32x32
未能转换 java.beans.BeanInfo.ICON_MONO_16x16
未能转换 java.beans.BeanInfo.ICON_MONO_32x32
未能转换 java.beans.Beans
未能转换 java.beans.Customizer
未能转换 java.beans.DesignMode
未能转换 java.beans.EventSetDescriptor
未能转换 java.beans.EventSetDescriptor.EventSetDescriptor
未能转换 java.beans.EventSetDescriptor.getAddListenerMethod
未能转换 java.beans.EventSetDescriptor.getListenerMethodDescriptors
未能转换 java.beans.EventSetDescriptor.getListenerMethods
未能转换 java.beans.EventSetDescriptor.getListenerType
未能转换 java.beans.EventSetDescriptor.getRemoveListenerMethod
未能转换 java.beans.EventSetDescriptor.isInDefaultEventSet
未能转换 java.beans.EventSetDescriptor.isUnicast
未能转换 java.beans.EventSetDescriptor.setInDefaultEventSet
未能转换 java.beans.EventSetDescriptor.setUnicast
未能转换 java.beans.FeatureDescriptor
未能转换 java.beans.FeatureDescriptor.attributeNames
未能转换 java.beans.FeatureDescriptor.FeatureDescriptor
未能转换 java.beans.FeatureDescriptor.getDisplayName
未能转换 java.beans.FeatureDescriptor.getName
未能转换 java.beans.FeatureDescriptor.getShortDescription
未能转换 java.beans.FeatureDescriptor.getValue
未能转换 java.beans.FeatureDescriptor.isExpert
未能转换 java.beans.FeatureDescriptor.isHidden
未能转换 java.beans.FeatureDescriptor.isPreferred
未能转换 java.beans.FeatureDescriptor.setDisplayName
未能转换 java.beans.FeatureDescriptor.setExpert
未能转换 java.beans.FeatureDescriptor.setHidden
未能转换 java.beans.FeatureDescriptor.setName

未能转换 java.beans.FeatureDescriptor.setPreferred
未能转换 java.beans.FeatureDescriptor.setShortDescription
未能转换 java.beans.FeatureDescriptor.setValue
未能转换 java.beans.IndexedPropertyDescriptor
未能转换 java.beans.IntrospectionException
未能转换 java.beans.Introspector
未能转换 java.beans.MethodDescriptor
未能转换 java.beans.ParameterDescriptor
未能转换 java.beans.PropertyChangeEvent.getPropagationId
未能转换 java.beans.PropertyChangeEvent.PropertyChangeEvent
未能转换 java.beans.PropertyChangeEvent.setPropagationId
未能转换 java.beans.PropertyChangeListener
未能转换 java.beans.PropertyChangeSupport
未能转换 java.beans.PropertyChangeSupport.addPropertyChangeListener
未能转换 java.beans.PropertyChangeSupport.firePropertyChange
未能转换 java.beans.PropertyChangeSupport.hasListeners
未能转换 java.beans.PropertyChangeSupport.PropertyChangeSupport
未能转换 java.beans.PropertyChangeSupport.removePropertyChangeListener
未能转换 java.beans.PropertyDescriptor
未能转换 java.beans.PropertyDescriptor.getPropertyEditorClass
未能转换 java.beans.PropertyDescriptor.getReadMethod
未能转换 java.beans.PropertyDescriptor.getWriteMethod
未能转换 java.beans.PropertyDescriptor.isBound
未能转换 java.beans.PropertyDescriptor.isConstrained
未能转换 java.beans.PropertyDescriptor.PropertyDescriptor
未能转换 java.beans.PropertyDescriptor.setBound
未能转换 java.beans.PropertyDescriptor.setConstrained
未能转换 java.beans.PropertyDescriptor.setPropertyEditorClass
未能转换 java.beans.PropertyDescriptor.setReadMethod
未能转换 java.beans.PropertyDescriptor.setWriteMethod
未能转换 java.beans.PropertyEditor
未能转换 java.beans.PropertyEditorManager
未能转换 java.beans.PropertyEditorSupport
未能转换 java.beans.PropertyVetoException.getPropertyChangeEvent
未能转换 java.beans.SimpleBeanInfo
未能转换 java.beans.VetoableChangeListener
未能转换 java.beans.VetoableChangeSupport
未能转换 java.beans.VetoableChangeSupport.addVetoableChangeListener
未能转换 java.beans.VetoableChangeSupport.fireVetoableChange
未能转换 java.beans.VetoableChangeSupport.hasListeners

未能转换 java.beans.VetoableChangeSupport.removeVetoableChangeListener

未能转换 java.beans.VetoableChangeSupport.VetoableChangeSupport

未能转换 java.beans.Visibility

Java.io 错误信息

未能转换 java.io.BufferedInputStream.available
未能转换 java.io.BufferedInputStream.buf
未能转换 java.io.BufferedInputStream.marklimit
未能转换 java.io.BufferedInputStream.markpos
未能转换 java.io.BufferedOutputStream.buf
未能转换 java.io.BufferedReader.BufferedReader
未能转换 java.io.BufferedReader.mark
未能转换 java.io.BufferedReader.reset
未能转换 java.io.BufferedWriter.BufferedWriter
未能转换 java.io.BufferedWriter.write
未能转换 java.io.ByteArrayInputStream.buf
未能转换 java.io.ByteArrayInputStream.mark~
未能转换 java.io.ByteArrayOutputStream.buf
未能转换 java.io.ByteArrayOutputStream.reset
未能转换 java.io.ByteArrayOutputStream.toString
未能转换 java.io.CharArrayReader.buf
未能转换 java.io.CharArrayReader.count
未能转换 java.io.CharArrayReader.mark
未能转换 java.io.CharArrayReader.markedPos
未能转换 java.io.CharArrayReader.pos
未能转换 java.io.CharArrayReader.reset
未能转换 java.io.CharArrayReader.skip
未能转换 java.io.CharArrayWriter.reset
未能转换 java.io.CharArrayWriter.size
未能转换 java.io.CharArrayWriter.toCharArray
未能转换 java.io.CharArrayWriter.toString
未能转换 java.io.CharArrayWriter.write
未能转换 java.io.CharArrayWriter.writeTo
未能转换 java.io.DataInput
未能转换 java.io.DataInput.readLine
未能转换 java.io.DataInput.readUTF
未能转换 java.io.DataInputStream
未能转换 java.io.DataInputStream.readLine
未能转换 java.io.DataInputStream.readUTF
未能转换 java.io.DataOutput
未能转换 java.io.DataOutput.writeBytes
未能转换 java.io.DataOutput.writeChars

未能转换 java.io.DataOutput.writeUTF
未能转换 java.io.DataOutputStream
未能转换 java.io.DataOutputStream.writeBytes
未能转换 java.io.DataOutputStream.writeChars
未能转换 java.io.DataOutputStream.writeUTF
未能转换 java.io.DataOutputStream.written
未能转换 java.io.Externalizable
未能转换 java.io.File.canRead
未能转换 java.io.File.createTempFile(String, String)
未能转换 java.io.File.createTempFile(String, String, File)
未能转换 java.io.File.deleteOnExit
未能转换 java.io.File.isAbsolute
未能转换 java.io.File.list
未能转换 java.io.File.listFiles
未能转换 java.io.File.mkdir
未能转换 java.io.File.mkdirs
未能转换 java.io.File.renameTo
未能转换 java.io.File.setLastModified
未能转换 java.io.FileDescriptor
未能转换 java.io.FileFilter
未能转换 java.io.FileInputStream.available
未能转换 java.io.FileInputStream.FileInputStream
未能转换 java.io.FileInputStream.getFD
未能转换 java.io.FilenameFilter
未能转换 java.io.FileOutputStream.FileOutputStream
未能转换 java.io.FileOutputStream.getFD
未能转换 java.io.FileOutputStream.write
未能转换 java.io.FilePermission.FilePermission
未能转换 java.io.FileReader.FileReader
未能转换 java.io.FileWriter.FileWriter
未能转换 java.io.FilterInputStream.available
未能转换 java.io.FilterInputStream.close
未能转换 java.io.FilterInputStream.FilterInputStream
未能转换 java.io.FilterInputStream.mark
未能转换 java.io.FilterInputStream.markSupported
未能转换 java.io.FilterInputStream.reset
未能转换 java.io.FilterReader.FilterReader
未能转换 java.io.FilterReader.in
未能转换 java.io.FilterReader.mark
未能转换 java.io.FilterReader.markSupported

未能转换 java.io.FilterReader.reset
未能转换 java.io.FileWriter
未能转换 java.io.FileWriter.getWriter
未能转换 java.io.FilterWriter.write
未能转换 java.io.InputStream.available
未能转换 java.io.InputStream.InputStream
未能转换 java.io.InputStream.mark
未能转换 java.io.InputStream.markSupported
未能转换 java.io.InputStream.read
未能转换 java.io.InputStream.reset
未能转换 java.io.InputStreamReader.InputStreamReader
未能转换 java.io.InputStreamReader.read
未能转换 java.io.InputStreamReader.read(char[], int, int)
未能转换 java.io.InterruptedIOException.bytesTransferred
未能转换 java.io.InvalidClassException.classname
未能转换 java.io.InvalidClassException.InvalidClassException
未能转换 java.io.LineNumberInputStream.available
未能转换 java.io.LineNumberInputStream.getLineNumber
未能转换 java.io.LineNumberInputStream.mark
未能转换 java.io.LineNumberInputStream.reset
未能转换 java.io.LineNumberInputStream.setLineNumber
未能转换 java.io.LineNumberReader.getLineNumber
未能转换 java.io.LineNumberReader.LineNumberReader
未能转换 java.io.LineNumberReader.mark
未能转换 java.io.LineNumberReader.reset
未能转换 java.io.LineNumberReader.setLineNumber
未能转换 java.io.ObjectInput
未能转换 java.io.ObjectInput.available
未能转换 java.io.ObjectInput.readObject
未能转换 java.io.ObjectInputStream
未能转换 java.io.ObjectInputStream.available
未能转换 java.io.ObjectInputStream.defaultReadObject
未能转换 java.io.ObjectInputStream.enableResolveObject
未能转换 java.io.ObjectInputStream.GetField
未能转换 java.io.ObjectInputStream.ObjectInputStream
未能转换 java.io.ObjectInputStream.readClassDescriptor
未能转换 java.io.ObjectInputStream.readFields
未能转换 java.io.ObjectInputStream.readLine
未能转换 java.io.ObjectInputStream.readObjectOverride
未能转换 java.io.ObjectInputStream.readStreamHeader

未能转换 java.io.ObjectInputStream.readUTF
未能转换 java.io.ObjectInputStream.registerValidation
未能转换 java.io.ObjectInputStream.resolveClass
未能转换 java.io.ObjectInputStream.resolveObject
未能转换 java.io.ObjectInputStream.resolveProxyClass
未能转换 java.io.ObjectInputStreamValidation
未能转换 java.io.ObjectOutput
未能转换 java.io.ObjectOutput.writeObject
未能转换 java.io.ObjectOutputStream
未能转换 java.io.ObjectOutputStream.annotateClass
未能转换 java.io.ObjectOutputStream.annotateProxyClass
未能转换 java.io.ObjectOutputStream.baseWireHandle
未能转换 java.io.ObjectOutputStream.defaultWriteObject
未能转换 java.io.ObjectOutputStream.enableReplaceObject
未能转换 java.io.ObjectOutputStream.ObjectOutputStream
未能转换 java.io.ObjectOutputStream.PutField
未能转换 java.io.ObjectOutputStream.putFields
未能转换 java.io.ObjectOutputStream.replaceObject
未能转换 java.io.ObjectOutputStream.reset
未能转换 java.io.ObjectOutputStream.SC_EXTERNALIZABLE
未能转换 java.io.ObjectOutputStream.SC_SERIALIZABLE
未能转换 java.io.ObjectOutputStream.SC_WRITE_METHOD
未能转换 java.io.ObjectOutputStream.useProtocolVersion
未能转换 java.io.ObjectOutputStream.writeBytes
未能转换 java.io.ObjectOutputStream.writeChars
未能转换 java.io.ObjectOutputStream.writeClassDescriptor
未能转换 java.io.ObjectOutputStream.writeFields
未能转换 java.io.ObjectOutputStream.writeObject
未能转换 java.io.ObjectOutputStream.writeObjectOverride
未能转换 java.io.ObjectOutputStream.writeStreamHeader
未能转换 java.io.ObjectOutputStream.writeUTF
未能转换 java.io.ObjectStreamClass
未能转换 java.io.OptionalDataException
未能转换 java.io.OutputStreamWriter.getEncoding
未能转换 java.io.OutputStreamWriter.OutputStreamWriter
未能转换 java.io.OutputStreamWriter.write
未能转换 java.io.PipedInputStream.available
未能转换 java.io.PipedInputStream.buffer
未能转换 java.io.PipedInputStream.connect
未能转换 java.io.PipedInputStream.in

未能转换 java.io.PipedInputStream.out
未能转换 java.io.PipedInputStream.PIPE_SIZE
未能转换 java.io.PipedInputStream.PipedInputStream
未能转换 java.io.PipedInputStream.read
未能转换 java.io.PipedInputStream.receive
未能转换 java.io.PipedOutputStream.connect
未能转换 java.io.PipedOutputStream.PipedOutputStream
未能转换 java.io.PipedOutputStream.write
未能转换 java.io.PipedReader.connect
未能转换 java.io.PipedWriter.connect
未能转换 java.io.PipedWriter.write
未能转换 java.io.PrintStream.checkError
未能转换 java.io.PrintStream.print
未能转换 java.io.PrintStream.println
未能转换 java.io.PrintStream setError
未能转换 java.io.PrintStream.write
未能转换 java.io.PrintWriter.checkError
未能转换 java.io.PrintWriter.print
未能转换 java.io.PrintWriter.PrintWriter
未能转换 java.io.PrintWriter.setError
未能转换 java.io.PrintWriter.write
未能转换 java.io.PushbackInputStream.available
未能转换 java.io.RandomAccessFile
未能转换 java.io.RandomAccessFile.getFD
未能转换 java.io.RandomAccessFile.RandomAccessFile(File, String)
未能转换 java.io.RandomAccessFile.RandomAccessFile(File, String, String)
未能转换 java.io.RandomAccessFile.readUTF
未能转换 java.io.RandomAccessFile.writeUTF
未能转换 java.io.Reader.lock
未能转换 java.io.Reader.mark
未能转换 java.io.Reader.markSupported
未能转换 java.io.Reader.read
未能转换 java.io.Reader.Reader
未能转换 java.io.Reader.reset
未能转换 java.io.SequenceInputStream.available
未能转换 java.io.SequenceInputStream.SequenceInputStream
未能转换 java.io.SerializablePermission
未能转换 java.io.StreamTokenizer
未能转换 java.io.StreamTokenizer.StreamTokenizer
未能转换 java.io.StringBufferInputStream.available

未能转换 java.io.StringBufferInputStream.buffer
未能转换 java.io.StringBufferInputStream.count
未能转换 java.io.StringBufferInputStream.pos
未能转换 java.io.StringBufferInputStream.read
未能转换 java.io.StringBufferInputStream.reset
未能转换 java.io.StringBufferInputStream.skip
未能转换 java.io.StringReader.mark
未能转换 java.io.StringReader.markSupported
未能转换 java.io.StringReader.ready
未能转换 java.io.StringReader.reset
未能转换 java.io.StringReader.skip
未能转换 java.io.StringWriter.getBuffer
未能转换 java.io.StringWriter.write
未能转换 java.io.Writer.lock
未能转换 java.io.Writer.write

Java.lang 错误信息

未能转换 java.lang.Boolean.getBoolean
未能转换 java.lang.Character.forDigit
未能转换 java.lang.Character.isDefined
未能转换 java.lang.Character.isIdentifierIgnorable
未能转换 java.lang.Character.isJavaIdentifierPart
未能转换 java.lang.Character.isUnicodeIdentifierPart
未能转换 java.lang.Character.isUnicodeIdentifierStart
未能转换 java.lang.Class.forName
未能转换 java.lang.Class.getClassLoader
未能转换 java.lang.Class.getDeclaredMethod
未能转换 java.lang.Class.getModifiers
未能转换 java.lang.Class.getPackage
未能转换 java.lang.Class.getProtectionDomain
未能转换 java.lang.Class.getResource
未能转换 java.lang.Class.getResourceAsStream
未能转换 java.lang.Class.getSigners
未能转换 java.lang.Class.newInstance
未能转换 java.lang.ClassLoader
未能转换 java.lang.ClassNotFoundException.printStackTrace
未能转换 java.lang.Compiler
未能转换 java.lang.Double.doubleToLongBits
未能转换 java.lang.Double.doubleToRawLongBits
未能转换 java.lang.Double.longBitsToDouble
未能转换 java.lang.ExceptionInInitializerError.getException
未能转换 java.lang.ExceptionInInitializerError.printStackTrace
未能转换 java.lang.Float.floatToIntBits
未能转换 java.lang.Float.floatToRawIntBits
未能转换 java.lang.Float.intBitsToFloat
未能转换 java.lang.InheritableThreadLocal
未能转换 java.lang.Integer.getInteger
未能转换 java.lang.Integer.TYPE
未能转换 java.lang.Long.getLong
未能转换 java.lang.Math.round
未能转换 java.lang.Number
未能转换 java.lang.Number.Number
未能转换 java.lang.Object.class
未能转换 java.lang.Object.clone

未能转换 java.lang.Package
未能转换 java.lang.ref.PhantomReference
未能转换 java.lang.ref.Reference
未能转换 java.lang.ref.ReferenceQueue
未能转换 java.lang.ref.SoftReference
未能转换 java.lang.ref.WeakReference.WakeReference
未能转换 java.lang.reflect.AccessibleObject
未能转换 java.lang.reflect.Constructor.getExceptionTypes
未能转换 java.lang.reflect.Constructor.newInstance
未能转换 java.lang.reflect.Field.getModifiers
未能转换 java.lang.reflect.Field.setByte
未能转换 java.lang.reflect.Field.setChar
未能转换 java.lang.reflect.Field.setShort
未能转换 java.lang.reflect.InvocationHandler
未能转换 java.lang.reflect.InvocationTargetException.InvocationTargetException
未能转换 java.lang.reflect.Member.DECLARED
未能转换 java.lang.reflect.Member.getModifiers
未能转换 java.lang.reflect.Member.PUBLIC
未能转换 java.lang.reflect.Method.getExceptionTypes
未能转换 java.lang.reflect.Method.getModifiers
未能转换 java.lang.reflect.Modifier
未能转换 java.lang.reflect.Proxy
未能转换 java.lang.reflect.ReflectPermission
未能转换 java.lang.Runtime.addShutdownHook
未能转换 java.lang.Runtime.exec
未能转换 java.lang.Runtime.freeMemory
未能转换 java.lang.Runtime.getLocalizedInputStream
未能转换 java.lang.Runtime.getLocalizedOutputStream
未能转换 java.lang.Runtime.halt
未能转换 java.lang.Runtime.load
未能转换 java.lang.Runtime.loadLibrary
未能转换 java.lang.Runtime.removeShutdownHook
未能转换 java.lang.Runtime.runFinalizersOnExit
未能转换 java.lang.Runtime.Runtime
未能转换 java.lang.Runtime.totalMemory
未能转换 java.lang.Runtime.traceInstructions
未能转换 java.lang.Runtime.traceMethodCalls
未能转换 java.lang.RuntimePermission
未能转换 java.lang.SecurityManager.checkAccess
未能转换 java.lang.SecurityManager.checkAwtEventQueueAccess

未能转换 java.lang.SecurityManager.checkConnect
未能转换 java.lang.SecurityManager.checkCreateClassLoader
未能转换 java.lang.SecurityManager.checkExec
未能转换 java.lang.SecurityManager.checkExit
未能转换 java.lang.SecurityManager.checkLink
未能转换 java.lang.SecurityManager.checkListen
未能转换 java.lang.SecurityManager.checkMemberAccess
未能转换 java.lang.SecurityManager.checkMulticast
未能转换 java.lang.SecurityManager.checkPackageAccess
未能转换 java.lang.SecurityManager.checkPackageDefinition
未能转换 java.lang.SecurityManager.checkPermission(Permission)
未能转换 java.lang.SecurityManager.checkPermission(Permission, Object)
未能转换 java.lang.SecurityManager.checkPrintJobAccess
未能转换 java.lang.SecurityManager.checkPropertiesAccess
未能转换 java.lang.SecurityManager.checkPropertyAccess
未能转换 java.lang.SecurityManager.checkRead
未能转换 java.lang.SecurityManager.checkSecurityAccess
未能转换 java.lang.SecurityManager.checkSetFactory
未能转换 java.lang.SecurityManager.checkTopLevelWindow
未能转换 java.lang.SecurityManager.checkWrite
未能转换 java.lang.SecurityManager.classDepth
未能转换 java.lang.SecurityManager.classLoaderDepth
未能转换 java.lang.SecurityManager.currentClassLoader
未能转换 java.lang.SecurityManager.currentLoadedClass
未能转换 java.lang.SecurityManager.getContext
未能转换 java.lang.SecurityManager.getInCheck
未能转换 java.lang.SecurityManager.getSecurityContext
未能转换 java.lang.SecurityManager.getThreadGroup
未能转换 java.lang.SecurityManager.inCheck
未能转换 java.lang.SecurityManager.inClass
未能转换 java.lang.SecurityManager.inClassLoader
未能转换 java.lang.SecurityManager.SecurityManager
未能转换 java.lang.StackOverflowError
未能转换 java.lang.StrictMath.round
未能转换 java.lang.String.CASE_INSENSITIVE_ORDER
未能转换 java.lang.String.getBytes
未能转换 java.lang.String.String(byte[])
未能转换 java.lang.String.String(byte[], int, int, int)
未能转换 java.lang.String.String(byte[], int, int, String)
未能转换 java.lang.String.String(byte[], String)

未能转换 java.lang.System
未能转换 java.lang.System.currentTimeMillis
未能转换 java.lang.System.getProperties
未能转换 java.lang.System.getProperty
未能转换 java.lang.System.getSecurityManager
未能转换 java.lang.System.load
未能转换 java.lang.System.loadLibrary
未能转换 java.lang.System.mapLibraryName
未能转换 java.lang.System.runFinalization
未能转换 java.lang.System.runFinalizersOnExit
未能转换 java.lang.System.setErr
未能转换 java.lang.System.setIn
未能转换 java.lang.System.setOut
未能转换 java.lang.System.setProperty
未能转换 java.lang.System.setProperties
未能转换 java.lang.System.setSecurityManager
未能转换 java.lang.Thread.activeCount
未能转换 java.lang.Thread.checkAccess
未能转换 java.lang.Thread.countStackFrames
未能转换 java.lang.Thread.destroy
未能转换 java.lang.Thread.enumerate
未能转换 java.lang.Thread.getContextClassLoader
未能转换 java.lang.Thread.getThreadGroup
未能转换 java.lang.Thread.interrupted
未能转换 java.lang.Thread.isInterrupted
未能转换 java.lang.Thread.run
未能转换 java.lang.Thread.setContextClassLoader
未能转换 java.lang.Thread.sleep(long)
未能转换 java.lang.Thread.sleep(long, int)
未能转换 java.lang.Thread.Thread
未能转换 java.lang.Thread.yield
未能转换 java.lang.ThreadGroup
未能转换 java.lang.ThreadLocal.initialValue
未能转换 java.lang.Throwable.fillInStackTrace
未能转换 java.lang.UnsupportedClassVersionError

Java.math 错误信息

未能转换 java.math.BigDecimal
未能转换 java.math.BigDecimal.BigDecimal
未能转换 java.math.BigDecimal.divide
未能转换 java.math.BigDecimal.movePointLeft
未能转换 java.math.BigDecimal.movePointRight
未能转换 java.math.BigDecimal.ROUND_CEILING
未能转换 java.math.BigDecimal.ROUND_DOWN
未能转换 java.math.BigDecimal.ROUND_FLOOR
未能转换 java.math.BigDecimal.ROUND_HALF_DOWN
未能转换 java.math.BigDecimal.ROUND_HALF_EVEN
未能转换 java.math.BigDecimal.ROUND_HALF_UP
未能转换 java.math.BigDecimal.ROUND_UNNECESSARY
未能转换 java.math.BigDecimal.ROUND_UP
未能转换 java.math.BigDecimal.scale
未能转换 java.math.BigDecimal.setScale
未能转换 java.math.BigDecimal.toBigInteger
未能转换 java.math.BigDecimal.unscaledValue
未能转换 java.math.BigDecimal.valueOf
未能转换 java.math.BigInteger
未能转换 java.math.BigInteger.and
未能转换 java.math.BigInteger.andNot
未能转换 java.math.BigInteger.BigInteger(byte[])
未能转换 java.math.BigInteger.BigInteger(int, byte[])
未能转换 java.math.BigInteger.BigInteger(int, int, Random)
未能转换 java.math.BigInteger.BigInteger(int, Random)
未能转换 java.math.BigInteger.BigInteger(string, int)
未能转换 java.math.BigInteger.bitCount
未能转换 java.math.BigInteger.bitLength
未能转换 java.math.BigInteger.clearBit
未能转换 java.math.BigInteger.flipBit
未能转换 java.math.BigInteger.gcd
未能转换 java.math.BigInteger.getLowestSetBit
未能转换 java.math.BigInteger.isProbablePrime
未能转换 java.math.BigInteger.modInverse
未能转换 java.math.BigInteger.modPow
未能转换 java.math.BigInteger.not
未能转换 java.math.BigInteger.or

未能转换 java.math.BigInteger.pow
未能转换 java.math.BigInteger.setBit
未能转换 java.math.BigInteger.shiftLeft
未能转换 java.math.BigInteger.shiftRight
未能转换 java.math.BigInteger.testBit
未能转换 java.math.BigInteger.toByteArray
未能转换 java.math.BigInteger.toString
未能转换 java.math.BigInteger.xor

Java.net 错误信息

未能转换 java.net.Authenticator
未能转换 java.net.ContentHandler
未能转换 java.net.ContentHandlerFactory
未能转换 java.net.DatagramPacket.DatagramPacket
未能转换 java.net.DatagramPacket.getOffset
未能转换 java.net DatagramSocket.getLocalAddress
未能转换 java.net DatagramSocket.getLocalPort
未能转换 java.net DatagramSocket.getReceiveBufferSize
未能转换 java.net DatagramSocket.getSendBufferSize
未能转换 java.net DatagramSocket.getSoTimeout
未能转换 java.net DatagramSocket.setDatagramSocketImplFactory
未能转换 java.net DatagramSocket.setReceiveBufferSize
未能转换 java.net DatagramSocket.setSendBufferSize
未能转换 java.net DatagramSocket.setSoTimeout
未能转换 java.net DatagramSocketImpl
未能转换 java.net DatagramSocketImplFactory
未能转换 java.net.FileNameMap
未能转换 java.net HttpURLConnection.getErrorStream
未能转换 java.net HttpURLConnection.getFollowRedirects
未能转换 java.net HttpURLConnection.getPermission
未能转换 java.net HttpURLConnection.getResponseCode
未能转换 java.net HttpURLConnection.getResponseMessage
未能转换 java.net HttpURLConnection.HTTP_MOVED_TEMP
未能转换 java.net HttpURLConnection.responseCode
未能转换 java.net HttpURLConnection.responseMessage
未能转换 java.net HttpURLConnection.usingProxy
未能转换 java.net.InetAddress.getAddress
未能转换 java.net.InetAddress.getAllByName
未能转换 java.net.InetAddress.getByName
未能转换 java.net.InetAddress.getLocalHost
未能转换 java.net.InetAddress.isMulticastAddress
未能转换 java.net.JarURLConnection
未能转换 java.net.MulticastSocket.getInterface
未能转换 java.net.MulticastSocket.getTimeToLive
未能转换 java.net.MulticastSocket.getTTL
未能转换 java.net.MulticastSocket.send
未能转换 java.net.MulticastSocket.setInterface

未能转换 java.net.MulticastSocket.setTimeToLive
未能转换 java.net.MulticastSocket.setTTL
未能转换 java.net.NetPermission.NetPermission
未能转换 java.net.NoRouteToHostException.NoRouteToHostException
未能转换 java.net.PlainDatagramSocketImpl
未能转换 java.net.PlainSocketImpl
未能转换 java.net.ServerSocket.getSoTimeout
未能转换 java.net.ServerSocket.implAccept
未能转换 java.net.ServerSocket.ServerSocket
未能转换 java.net.ServerSocket.setSocketFactory
未能转换 java.net.ServerSocket.setSoTimeout
未能转换 java.net.Socket.getInetAddress
未能转换 java.net.Socket.getKeepAlive
未能转换 java.net.Socket.getLocalAddress
未能转换 java.net.Socket.getLocalPort
未能转换 java.net.Socket.getPort
未能转换 java.net.Socket.setKeepAlive
未能转换 java.net.Socket.setSocketImplFactory
未能转换 java.net.Socket.shutdownInput
未能转换 java.net.Socket.shutdownOutput
未能转换 java.net.Socket.Socket
未能转换 java.net.SocketException.SocketException
未能转换 java.net.SocketImpl
未能转换 java.net.SocketImplFactory
未能转换 java.net.SocketInputStream
未能转换 java.net.SocketOptions
未能转换 java.net.SocketOutputStream
未能转换 java.net.SocketPermission.SocketPermission
未能转换 java.net.UnknownContentHandler
未能转换 java.net.URL.getContent
未能转换 java.net.URL.getContent(Class[])
未能转换 java.net.URL.getPort
未能转换 java.net.URL.openStream
未能转换 java.net.URL.set
未能转换 java.net.URL.setURLStreamHandlerFactory
未能转换 java.net.URL.URL
未能转换 java.net.URLClassLoader
未能转换 java.netURLConnection.allowUserInteraction
未能转换 java.netURLConnection.connect
未能转换 java.netURLConnection.doInput

未能转换 java.net.URLConnection.doOutput
未能转换 java.net.URLConnection.fileNameMap
未能转换 java.net.URLConnection.getAllowUserInteraction
未能转换 java.net.URLConnection.getContent
未能转换 java.net.URLConnection.getContentEncoding
未能转换 java.net.URLConnection.getDate
未能转换 java.net.URLConnection.getDefaultAllowUserInteraction
未能转换 java.net.URLConnection.getDefaultRequestProperty
未能转换 java.net.URLConnection.getDefaultUseCaches
未能转换 java.net.URLConnection.getDoInput
未能转换 java.net.URLConnection.getDoOutput
未能转换 java.net.URLConnection.getFileNameMap
未能转换 java.net.URLConnection.getHeaderField
未能转换 java.net.URLConnection.getHeaderFieldKey
未能转换 java.net.URLConnection.getIfModifiedSince
未能转换 java.net.URLConnection.getLastModified
未能转换 java.net.URLConnection.getPermission
未能转换 java.net.URLConnection.getRequestProperty
未能转换 java.net.URLConnection.getUseCaches
未能转换 java.net.URLConnection.guessContentTypeFromName
未能转换 java.net.URLConnection.guessContentTypeFromStream
未能转换 java.net.URLConnection.setAllowUserInteraction
未能转换 java.net.URLConnection.setContentHandlerFactory
未能转换 java.net.URLConnection.setDefaultAllowUserInteraction
未能转换 java.net.URLConnection.setDefaultRequestProperty
未能转换 java.net.URLConnection.setDefaultUseCaches
未能转换 java.net.URLConnection.setDoInput
未能转换 java.net.URLConnection.setDoOutput
未能转换 java.net.URLConnection.setFileNameMap
未能转换 java.net.URLConnection.setUseCaches
未能转换 java.net.URLConnection.toString
未能转换 java.net.URLConnection.url
未能转换 java.netURLConnection.useCaches
未能转换 java.net.URLDecoder
未能转换 java.net.URLEncoder
未能转换 java.net.URLStreamHandler
未能转换 java.net.URLStreamHandlerFactory

Java.rmi 错误信息

未能转换 java.rmi.AccessException
未能转换 java.rmi.activation.ActivatableActivatable
未能转换 java.rmi.activation.Activatable.exportObject
未能转换 java.rmi.activation.Activatable.getID
未能转换 java.rmi.activation.Activatable.inactive
未能转换 java.rmi.activation.Activatable.register
未能转换 java.rmi.activation.Activatable.unexportObject
未能转换 java.rmi.activation.Activatable.unregister
未能转换 java.rmi.activation.ActivateFailedException
未能转换 java.rmi.activation.ActivationDesc
未能转换 java.rmi.activation.ActivationException
未能转换 java.rmi.activation.ActivationGroup
未能转换 java.rmi.activation.ActivationGroup_Stub
未能转换 java.rmi.activation.ActivationGroupDesc
未能转换 java.rmi.activation.ActivationGroupDesc.CommandEnvironment
未能转换 java.rmi.activation.ActivationGroupID
未能转换 java.rmi.activation.ActivationID
未能转换 java.rmi.activation.ActivationInstantiator
未能转换 java.rmi.activation.ActivationMonitor
未能转换 java.rmi.activation.ActivationSystem
未能转换 java.rmi.activation.Activator.activate
未能转换 java.rmi.activation.CommandEnvironment.CommandEnvironment
未能转换 java.rmi.activation.CommandEnvironment.equals
未能转换 java.rmi.activation.CommandEnvironment.getCommandOptions
未能转换 java.rmi.activation.CommandEnvironment.getCommandPath
未能转换 java.rmi.activation.CommandEnvironment.hashCode
未能转换 java.rmi.activation.UnknownGroupException
未能转换 java.rmi.activation.UnknownObjectException
未能转换 java.rmi.AlreadyBoundException
未能转换 java.rmi.ConnectException
未能转换 java.rmi.ConnectIOException
未能转换 java.rmi.dgc.DGC
未能转换 java.rmi.dgc.Lease.getVMID
未能转换 java.rmi.dgc.VMID
未能转换 java.rmi.MarshalException
未能转换 java.rmi.MarshalledObject
未能转换 java.rmi.Naming

未能转换 java.rmi.Naming.bind
未能转换 java.rmi.Naming.lookup
未能转换 java.rmi.Naming.rebind
未能转换 java.rmi.Naming.unbind
未能转换 java.rmi.NoSuchObjectException
未能转换 java.rmi.NotBoundException
未能转换 java.rmi.registry.LocateRegistry
未能转换 java.rmi.registry.Registry
未能转换 java.rmi.registry.Registry.bind
未能转换 java.rmi.registry.Registry.lookup
未能转换 java.rmi.registry.Registry.rebind
未能转换 java.rmi.registry.Registry.REGISTRY_PORT
未能转换 java.rmi.registry.Registry.unbind
未能转换 java.rmi.registry.RegistryHandler
未能转换 java.rmi.RemoteException.detail
未能转换 java.rmi.RMISecurityException
未能转换 java.rmi.RMISecurityManager
未能转换 java.rmi.RMISecurityManager.RMISecurityManager
未能转换 java.rmi.server.ExportException
未能转换 java.rmi.server.LoaderHandler
未能转换 java.rmi.server.LogStream
未能转换 java.rmi.server.ObjID
未能转换 java.rmi.server.Operation
未能转换 java.rmi.server.RemoteCall
未能转换 java.rmi.server.RemoteRef
未能转换 java.rmi.server.RemoteServer
未能转换 java.rmi.server.RemoteStub
未能转换 java.rmi.server.RMIClassLoader
未能转换 java.rmi.server.RMIFailureHandler
未能转换 java.rmi.server.RMISocketFactory.getDefaultSocketFactory
未能转换 java.rmi.server.RMISocketFactory.getFailureHandler
未能转换 java.rmi.server.RMISocketFactory.getSocketFactory
未能转换 java.rmi.server.RMISocketFactory.setFailureHandler
未能转换 java.rmi.server.RMISocketFactory.setSocketFactory
未能转换 java.rmi.server.RemoteObject.getRef
未能转换 java.rmi.server.RemoteObject.ref
未能转换 java.rmi.server.RemoteObject.RemoteObject
未能转换 java.rmi.server.RemoteObject.toStub
未能转换 java.rmi.server.RemoteServer.getClientHost
未能转换 java.rmi.server.RemoteServer.getLog

未能转换 java.rmi.server.RemoteServer.RemoteServer
未能转换 java.rmi.server.ServerCloneException
未能转换 java.rmi.server.ServerNotActiveException
未能转换 java.rmi.server.ServerRef
未能转换 java.rmi.server.Skeleton
未能转换 java.rmi.server.SkeletonMismatchException
未能转换 java.rmi.server.SkeletonNotFoundException
未能转换 java.rmi.server.SocketSecurityException
未能转换 java.rmi.server.UID
未能转换 java.rmi.server.UnicastRemoteObject.clone
未能转换 java.rmi.server.UnicastRemoteObject.exportObject
未能转换 java.rmi.server.UnicastRemoteObject.unexportObject
未能转换 java.rmi.server.UnicastRemoteObject.UnicastRemoteObject
未能转换 java.rmi.server.Unreferenced
未能转换 java.rmi.ServerRuntimeException
未能转换 java.rmi.StubNotFoundException
未能转换 java.rmi.UnexpectedException
未能转换 java.rmi.UnknownHostException
未能转换 java.rmi.UnmarshalException

Java.security 错误信息

未能转换 java.security.AccessControlContext
未能转换 java.security.AccessControlException.getPermission
未能转换 java.security.AccessController
未能转换 java.security.acl
未能转换 java.security.acl.Acl
未能转换 java.security.acl.AclEntry
未能转换 java.security.acl.AclNotFoundException
未能转换 java.security.acl.Group
未能转换 java.security.acl.LastOwnerException
未能转换 java.security.acl.NotOwnerException
未能转换 java.security.acl.Owner
未能转换 java.security.acl.Permission
未能转换 java.security.AlgorithmParameterGenerator
未能转换 java.security.AlgorithmParameterGeneratorSpi
未能转换 java.security.AlgorithmParameters
未能转换 java.security.AlgorithmParametersSpi
未能转换 java.security.AllPermission
未能转换 java.security.BasicPermission.newPermissionCollection
未能转换 java.security.cert.Certificate.Certificate
未能转换 java.security.cert.Certificate.getEncoded
未能转换 java.security.cert.Certificate.verify(PublicKey)
未能转换 java.security.cert.Certificate.verify(PublicKey, String)
未能转换 java.security.cert.Certificate.writeReplace
未能转换 java.security.cert.CertificateFactory.CertificateFactory
未能转换 java.security.cert.CertificateFactory.generateCertificate
未能转换 java.security.cert.CertificateFactory.generateCertificates
未能转换 java.security.cert.CertificateFactory.generateCRL
未能转换 java.security.cert.CertificateFactory.generateCRLs
未能转换 java.security.cert.CertificateFactory.getInstance
未能转换 java.security.cert.CertificateFactory.getProvider
未能转换 java.security.cert.CertificateFactorySpi
未能转换 java.security.cert.CRL
未能转换 java.security.cert.X509Certificate.getBasicConstraints
未能转换 java.security.cert.X509Certificate.getCriticalExtensionOIDs
未能转换 java.security.cert.X509CertificategetExtensionValue
未能转换 java.security.cert.X509Certificate.getIssuerUniqueID
未能转换 java.security.cert.X509Certificate.getKeyUsage

未能转换 java.security.cert.X509Certificate.getNonCriticalExtensionOIDs
未能转换 java.security.cert.X509Certificate.getNotAfter
未能转换 java.security.cert.X509Certificate.getNotBefore
未能转换 java.security.cert.X509CRLEntry.getRevocationDate
未能转换 java.security.cert.X509Certificate.getSigAlgName
未能转换 java.security.cert.X509Certificate.getSignature
未能转换 java.security.cert.X509Certificate.getSubjectUniqueId
未能转换 java.security.cert.X509Certificate.getTBS Certificate
未能转换 java.security.cert.X509Certificate.getVersion
未能转换 java.security.cert.X509Certificate.hasUnsupportedCriticalExtension
未能转换 java.security.cert.X509CRL
未能转换 java.security.cert.X509CRLEntry.getCriticalExtensionOIDs
未能转换 java.security.cert.X509CRLEntry.getEncoded
未能转换 java.security.cert.X509CRLEntrygetExtensionValue
未能转换 java.security.cert.X509CRLEntry.getNonCriticalExtensionOIDs
未能转换 java.security.cert.X509CRLEntry.hasExtensions
未能转换 java.security.cert.X509CRLEntry.hasUnsupportedCriticalExtension
未能转换 java.security.cert.X509Extension
未能转换 java.security.Certificate
未能转换 java.security.CodeSource
未能转换 java.security.DomainCombiner
未能转换 java.security.Guard
未能转换 java.security.GuardedObject
未能转换 java.security.Identity
未能转换 java.security.IdentityScope
未能转换 java.security.interfaces.DSAKeyPairGenerator
未能转换 java.security.interfaces.DSAParams.getG
未能转换 java.security.interfaces.DSAParams.getP
未能转换 java.security.interfaces.DSAParams.getQ
未能转换 java.security.interfaces.DSAPrivateKey
未能转换 java.security.interfaces.DSAPublicKey
未能转换 java.security.interfaces.RSAPrivateCrtKey
未能转换 java.security.interfaces.RSAPrivateKey
未能转换 java.security.Key.getFormat
未能转换 java.security.KeyFactory
未能转换 java.security.KeyFactorySpi
未能转换 java.security.KeyPairGenerator
未能转换 java.security.KeyPairGeneratorSpi
未能转换 java.security.KeyStore

未能转换 java.security.KeyStoreSpi
未能转换 java.security.MessageDigest.clone
未能转换 java.security.MessageDigest.getInstance
未能转换 java.security.MessageDigestSpi.clone
未能转换 java.security.MessageDigestSpi.MessageDigestSpi
未能转换 java.security.Permission.checkGuard
未能转换 java.security.Permission.getName
未能转换 java.security.PermissionCollection.implies
未能转换 java.security.PermissionCollection.setReadOnly
未能转换 java.security.Policy
未能转换 java.security.ProtectionDomain
未能转换 java.security.Provider
未能转换 java.security.SecureClassLoader
未能转换 java.security.SecureRandom.getProvider
未能转换 java.security.SecureRandom.next
未能转换 java.security.SecureRandomSpi.SecureRandomSpi
未能转换 java.security.Security
未能转换 java.security.SecurityPermission.SecurityPermission
未能转换 java.security.Signature.clone
未能转换 java.security.Signature.engineGetParameter
未能转换 java.security.Signature.engineInitSign
未能转换 java.security.Signature.engineInitVerify
未能转换 java.security.Signature.engineSetParameter
未能转换 java.security.Signature.engineSign
未能转换 java.security.Signature.engineVerify
未能转换 java.security.Signature.getInstance
未能转换 java.security.Signature.getParameter
未能转换 java.security.Signature.initSign
未能转换 java.security.Signature.initVerify
未能转换 java.security.Signature.setParameter
未能转换 java.security.Signature.sign
未能转换 java.security.Signature.SIGN~
未能转换 java.security.Signature.state
未能转换 java.security.Signature.UNINITIALIZED
未能转换 java.security.Signature.verify
未能转换 java.security.Signature.VERIFY~
未能转换 java.security.SignatureSpi.appRandom
未能转换 java.security.SignatureSpi.clone
未能转换 java.security.SignatureSpi.engineGetParameter
未能转换 java.security.SignatureSpi.engineInitSign

未能转换 java.security.SignatureSpi.engineSetParameter
未能转换 java.security.SignatureSpi.engineSign
未能转换 java.security.SignatureSpi.engineSign(byte[], int, int)
未能转换 java.security.SignatureSpi.engineVerify
未能转换 java.security.SecureRandomSpi.SecureRandomSpi
未能转换 java.security.SignedObject
未能转换 java.security.Signer
未能转换 java.security.spec.DSAPrivateKeySpec.DSAPrivateKeySpec
未能转换 java.security.spec.DSAPublicKeySpec.DSAPublicKeySpec
未能转换 java.security.spec.DSAPublicKeySpec.DSAPublicKeySpec
未能转换 java.security.spec.EncodedKeySpec
未能转换 java.security.spec.PKCS8EncodedKeySpec
未能转换 java.security.spec.RSAKeyGenParameterSpec
未能转换 java.security.spec.RSAPrivateCrtKeySpec
未能转换 java.security.spec.RSAPrivateKeySpec.RSAPrivateKeySpec
未能转换 java.security.spec.RSAPublicKeySpec.RSAPublicKeySpec
未能转换 java.security.spec.X509EncodedKeySpec
未能转换 java.security.UnresolvedPermission

Java.sql 错误信息

未能转换 java.sql.Array
未能转换 java.sql.BatchUpdateException
未能转换 java.sql.Blob
未能转换 java.sql.Blob.getBinaryStream
未能转换 java.sql.Blob.getBytes
未能转换 java.sql.Blob.position
未能转换 java.sql.CallableStatement.getArray
未能转换 java.sql.CallableStatement.getBigDecimal
未能转换 java.sql.CallableStatement.getBlob
未能转换 java.sql.CallableStatement.getBlob
未能转换 java.sql.CallableStatement.getDate
未能转换 java.sql.CallableStatement.getObject
未能转换 java.sql.CallableStatement.getRef
未能转换 java.sql.CallableStatement.getTime
未能转换 java.sql.CallableStatement.getTimestamp
未能转换 java.sql.CallableStatement.registerOutParameter
未能转换 java.sql.CallableStatement.wasNull
未能转换 java.sql.Clob
未能转换 java.sql.Clob.getAsciiStream
未能转换 java.sql.Clob.getCharacterStream
未能转换 java.sql.Clob.getSubString
未能转换 java.sql.Clob.position
未能转换 java.sql.Connection.clearWarnings
未能转换 java.sql.Connection.createStatement
未能转换 java.sql.Connection.getTypeMap
未能转换 java.sql.Connection.getWarnings
未能转换 java.sql.Connection.isReadOnly
未能转换 java.sql.Connection.nativeSQL
未能转换 java.sql.Connection.prepareCall
未能转换 java.sql.Connection.prepareStatement
未能转换 java.sql.Connection.setCatalog
未能转换 java.sql.Connection.setReadOnly
未能转换 java.sql.Connection.setTypeMap
未能转换 java.sql.DatabaseMetaData.allProceduresAreCallable
未能转换 java.sql.DatabaseMetaData.allTablesAreSelectable
未能转换 java.sql.DatabaseMetaData.bestRowNotPseudo
未能转换 java.sql.DatabaseMetaData.bestRowPseudo

未能转换 java.sql.DatabaseMetaData.bestRowSession
未能转换 java.sql.DatabaseMetaData.bestRowTemporary
未能转换 java.sql.DatabaseMetaData.bestRowTransaction
未能转换 java.sql.DatabaseMetaData.bestRowUnknown
未能转换 java.sql.DatabaseMetaData.columnNoNulls
未能转换 java.sql.DatabaseMetaData.columnNullable
未能转换 java.sql.DatabaseMetaData.columnNullableUnknown
未能转换 java.sql.DatabaseMetaData.dataDefinitionCausesTransactionCommit
未能转换 java.sql.DatabaseMetaData.dataDefinitionIgnoredInTransactions
未能转换 java.sql.DatabaseMetaData.deletesAreDetected
未能转换 java.sql.DatabaseMetaData.doesMaxRowSizeIncludeBlobs
未能转换 java.sql.DatabaseMetaData.getBestRowIdentifier
未能转换 java.sql.DatabaseMetaData.getCatalogTerm
未能转换 java.sql.DatabaseMetaData.getColumnPrivileges
未能转换 java.sql.DatabaseMetaData.getColumns
未能转换 java.sql.DatabaseMetaData.getCrossReference
未能转换 java.sql.DatabaseMetaData.getDatabaseProductName
未能转换 java.sql.DatabaseMetaData.getDriverMajorVersion
未能转换 java.sql.DatabaseMetaData.getDriverMinorVersion
未能转换 java.sql.DatabaseMetaData.getDriverName
未能转换 java.sql.DatabaseMetaData.getDriverVersion
未能转换 java.sql.DatabaseMetaData.getExportedKeys
未能转换 java.sql.DatabaseMetaData.getExtraNameCharacters
未能转换 java.sql.DatabaseMetaData.getIdentifierQuoteString
未能转换 java.sql.DatabaseMetaData.getImportedKeys
未能转换 java.sql.DatabaseMetaData.getIndexInfo
未能转换 java.sql.DatabaseMetaData.getMaxColumnsInGroupBy
未能转换 java.sql.DatabaseMetaData.getMaxColumnsInIndex
未能转换 java.sql.DatabaseMetaData.getMaxColumnsInOrderBy
未能转换 java.sql.DatabaseMetaData.getMaxColumnsInSelect
未能转换 java.sql.DatabaseMetaData.getMaxColumnsInTable
未能转换 java.sql.DatabaseMetaData.getMaxConnections
未能转换 java.sql.DatabaseMetaData.getMaxIndexLength
未能转换 java.sql.DatabaseMetaData.getMaxRowSize
未能转换 java.sql.DatabaseMetaData.getMaxStatementLength
未能转换 java.sql.DatabaseMetaData.getMaxStatements
未能转换 java.sql.DatabaseMetaData.getMaxTablesInSelect
未能转换 java.sql.DatabaseMetaData.getNumericFunctions
未能转换 java.sql.DatabaseMetaData.getPrimaryKeys
未能转换 java.sql.DatabaseMetaData.getProcedureColumns

未能转换 java.sql.DatabaseMetaData.getProcedures
未能转换 java.sql.DatabaseMetaData.getProcedureTerm
未能转换 java.sql.DatabaseMetaData.getSchemaTerm
未能转换 java.sql.DatabaseMetaData.getSearchStringEscape
未能转换 java.sql.DatabaseMetaData.getSQLKeywords
未能转换 java.sql.DatabaseMetaData.getStringFunctions
未能转换 java.sql.DatabaseMetaData.getSystemFunctions
未能转换 java.sql.DatabaseMetaData.getTablePrivileges
未能转换 java.sql.DatabaseMetaData.getTables
未能转换 java.sql.DatabaseMetaData.getTimeDateFunctions
未能转换 java.sql.DatabaseMetaData.getUDTs
未能转换 java.sql.DatabaseMetaData.getURL
未能转换 java.sql.DatabaseMetaData.getUserName
未能转换 java.sql.DatabaseMetaData.getVersionColumns
未能转换 java.sql.DatabaseMetaData.importedKeyCascade
未能转换 java.sql.DatabaseMetaData.importedKeyInitiallyDeferred
未能转换 java.sql.DatabaseMetaData.importedKeyInitiallyImmediate
未能转换 java.sql.DatabaseMetaData.importedKeyNoAction
未能转换 java.sql.DatabaseMetaData.importedKeyNotDeferrable
未能转换 java.sql.DatabaseMetaData.importedKeyRestrict
未能转换 java.sql.DatabaseMetaData.importedKeySetDefault
未能转换 java.sql.DatabaseMetaData.importedKeySetNull
未能转换 java.sql.DatabaseMetaData.insertsAreDetected
未能转换 java.sql.DatabaseMetaData.isCatalogAtStart
未能转换 java.sql.DatabaseMetaData.isReadOnly
未能转换 java.sql.DatabaseMetaData.nullPlusNonNullIsNull
未能转换 java.sql.DatabaseMetaData.nullsAreSortedAtEnd
未能转换 java.sql.DatabaseMetaData.nullsAreSortedAtStart
未能转换 java.sql.DatabaseMetaData.nullsAreSortedHigh
未能转换 java.sql.DatabaseMetaData.nullsAreSortedLow
未能转换 java.sql.DatabaseMetaData.othersDeletesAreVisible
未能转换 java.sql.DatabaseMetaData.othersInsertsAreVisible
未能转换 java.sql.DatabaseMetaData.othersUpdatesAreVisible
未能转换 java.sql.DatabaseMetaData.ownDeletesAreVisible
未能转换 java.sql.DatabaseMetaData.ownInsertsAreVisible
未能转换 java.sql.DatabaseMetaData.ownUpdatesAreVisible
未能转换 java.sql.DatabaseMetaData.procedureColumnIn
未能转换 java.sql.DatabaseMetaData.procedureColumnInOut
未能转换 java.sql.DatabaseMetaData.procedureColumnOut
未能转换 java.sql.DatabaseMetaData.procedureColumnResult

未能转换 java.sql.DatabaseMetaData.procedureColumnReturn
未能转换 java.sql.DatabaseMetaData.procedureColumnUnknown
未能转换 java.sql.DatabaseMetaData.procedureNoNulls
未能转换 java.sql.DatabaseMetaData.procedureNoResult
未能转换 java.sql.DatabaseMetaData.procedureNullable
未能转换 java.sql.DatabaseMetaData.procedureNullableUnknown
未能转换 java.sql.DatabaseMetaData.procedureResultUnknown
未能转换 java.sql.DatabaseMetaData.procedureReturnsResult
未能转换 java.sql.DatabaseMetaData.storesLowerCaselidentifiers
未能转换 java.sql.DatabaseMetaData.storesLowerCaseQuotedIdentifiers
未能转换 java.sql.DatabaseMetaData.storesMixedCaselidentifiers
未能转换 java.sql.DatabaseMetaData.storesMixedCaseQuotedIdentifiers
未能转换 java.sql.DatabaseMetaData.storesUpperCaselidentifiers
未能转换 java.sql.DatabaseMetaData.storesUpperCaseQuotedIdentifiers
未能转换 java.sql.DatabaseMetaData.supportsAlterTableWithAddColumn
未能转换 java.sql.DatabaseMetaData.supportsAlterTableWithDropColumn
未能转换 java.sql.DatabaseMetaData.supportsANSI92EntryLevelSQL
未能转换 java.sql.DatabaseMetaData.supportsANSI92FullSQL
未能转换 java.sql.DatabaseMetaData.supportsANSI92IntermediateSQL
未能转换 java.sql.DatabaseMetaData.supportsBatchUpdates
未能转换 java.sql.DatabaseMetaData.supportsCatalogsInDataManipulation
未能转换 java.sql.DatabaseMetaData.supportsCatalogsInIndexDefinitions
未能转换 java.sql.DatabaseMetaData.supportsCatalogsInPrivilegeDefinitions
未能转换 java.sql.DatabaseMetaData.supportsCatalogsInProcedureCalls
未能转换 java.sql.DatabaseMetaData.supportsCatalogsInTableDefinitions
未能转换 java.sql.DatabaseMetaData.supportsColumnAliasing
未能转换 java.sql.DatabaseMetaData.supportsConvert
未能转换 java.sql.DatabaseMetaData.supportsCoreSQLGrammar
未能转换 java.sql.DatabaseMetaData.supportsCorrelatedSubqueries
未能转换 java.sql.DatabaseMetaData.supportsDataDefinitionAndDataManipulationTransactions
未能转换 java.sql.DatabaseMetaData.supportsDataManipulationTransactionsOnly
未能转换 java.sql.DatabaseMetaData.supportsDifferentTableCorrelationNames
未能转换 java.sql.DatabaseMetaData.supportsExpressionsInOrderBy
未能转换 java.sql.DatabaseMetaData.supportsExtendedSQLGrammar
未能转换 java.sql.DatabaseMetaData.supportsFullOuterJoins
未能转换 java.sql.DatabaseMetaData.supportsGroupBy
未能转换 java.sql.DatabaseMetaData.supportsGroupByBeyondSelect
未能转换 java.sql.DatabaseMetaData.supportsGroupByUnrelated
未能转换 java.sql.DatabaseMetaData.supportsIntegrityEnhancementFacility
未能转换 java.sql.DatabaseMetaData.supportsLikeEscapeClause

未能转换 java.sql.DatabaseMetaData.supportsLimitedOuterJoins
未能转换 java.sql.DatabaseMetaData.supportsMinimumSQLGrammar
未能转换 java.sql.DatabaseMetaData.supportsMixedCaseIdentifiers
未能转换 java.sql.DatabaseMetaData.supportsMixedCaseQuotedIdentifiers
未能转换 java.sql.DatabaseMetaData.supportsMultipleResultSets
未能转换 java.sql.DatabaseMetaData.supportsMultipleTransactions
未能转换 java.sql.DatabaseMetaData.supportsNonNullableColumns
未能转换 java.sql.DatabaseMetaData.supportsOpenCursorsAcrossCommit
未能转换 java.sql.DatabaseMetaData.supportsOpenCursorsAcrossRollback
未能转换 java.sql.DatabaseMetaData.supportsOpenStatementsAcrossCommit
未能转换 java.sql.DatabaseMetaData.supportsOpenStatementsAcrossRollback
未能转换 java.sql.DatabaseMetaData.supportsOrderByUnrelated
未能转换 java.sql.DatabaseMetaData.supportsOuterJoins
未能转换 java.sql.DatabaseMetaData.supportsPositionedDelete
未能转换 java.sql.DatabaseMetaData.supportsPositionedUpdate
未能转换 java.sql.DatabaseMetaData.supportsResultSetConcurrency
未能转换 java.sql.DatabaseMetaData.supportsResultSetType
未能转换 java.sql.DatabaseMetaData.supportsSchemasInDataManipulation
未能转换 java.sql.DatabaseMetaData.supportsSchemasInIndexDefinitions
未能转换 java.sql.DatabaseMetaData.supportsSchemasInPrivilegeDefinitions
未能转换 java.sql.DatabaseMetaData.supportsSchemasInProcedureCalls
未能转换 java.sql.DatabaseMetaData.supportsSchemasInTableDefinitions
未能转换 java.sql.DatabaseMetaData.supportsSelectForUpdate
未能转换 java.sql.DatabaseMetaData.supportsStoredProcedures
未能转换 java.sql.DatabaseMetaData.supportsSubqueriesInComparisons
未能转换 java.sql.DatabaseMetaData.supportsSubqueriesInExists
未能转换 java.sql.DatabaseMetaData.supportsSubqueriesInIns
未能转换 java.sql.DatabaseMetaData.supportsSubqueriesInQuantifieds
未能转换 java.sql.DatabaseMetaData.supportsTableCorrelationNames
未能转换 java.sql.DatabaseMetaData.supportsTransactionIsolationLevel
未能转换 java.sql.DatabaseMetaData.supportsTransactions
未能转换 java.sql.DatabaseMetaData.supportsUnion
未能转换 java.sql.DatabaseMetaData.supportsUnionAll
未能转换 java.sql.DatabaseMetaData.tableIndexClustered
未能转换 java.sql.DatabaseMetaData.tableIndexHashed
未能转换 java.sql.DatabaseMetaData.tableIndexOther
未能转换 java.sql.DatabaseMetaData.tableIndexStatistic
未能转换 java.sql.DatabaseMetaData.typeNoNulls
未能转换 java.sql.DatabaseMetaData.typeNullable
未能转换 java.sql.DatabaseMetaData.typeNullableUnknown

未能转换 java.sql.DatabaseMetaData.typePredBasic
未能转换 java.sql.DatabaseMetaData.typePredChar
未能转换 java.sql.DatabaseMetaData.typePredNone
未能转换 java.sql.DatabaseMetaData.typeSearchable
未能转换 java.sql.DatabaseMetaData.updatesAreDetected
未能转换 java.sql.DatabaseMetaData.usesLocalFilePerTable
未能转换 java.sql.DatabaseMetaData.usesLocalFiles
未能转换 java.sql.DatabaseMetaData.versionColumnNotPseudo
未能转换 java.sql.DatabaseMetaData.versionColumnPseudo
未能转换 java.sql.DatabaseMetaData.versionColumnUnknown
未能转换 java.sql.DataTruncation
未能转换 java.sql.Date.Date
未能转换 java.sql.Date.getHours
未能转换 java.sql.Date.getMinutes
未能转换 java.sql.Date.getSeconds
未能转换 java.sql.Date.setHours
未能转换 java.sql.Date.setMinutes
未能转换 java.sql.Date.setSeconds
未能转换 java.sql.DriverInfo
未能转换 java.sql.DriverManager
未能转换 java.sql.DriverPropertyInfo
未能转换 java.sql.PreparedStatement.addBatch
未能转换 java.sql.PreparedStatement.getMetaData
未能转换 java.sql.PreparedStatement.setArray
未能转换 java.sql.PreparedStatement.setBlob
未能转换 java.sql.PreparedStatement.setClob
未能转换 java.sql.PreparedStatement.setDate
未能转换 java.sql.PreparedStatement.setNull
未能转换 java.sql.PreparedStatement.setRef
未能转换 java.sql.PreparedStatement.setTime
未能转换 java.sql.PreparedStatement.setTimestamp
未能转换 java.sql.Ref
未能转换 java.sql.ResultSet
未能转换 java.sql.ResultSet.<ConcurrencyMode>
未能转换 java.sql.ResultSet.absolute
未能转换 java.sql.ResultSet.afterLast
未能转换 java.sql.ResultSet.beforeFirst
未能转换 java.sql.ResultSet.cancelRowUpdates
未能转换 java.sql.ResultSet.clearWarnings
未能转换 java.sql.ResultSet.close

未能转换 java.sql.ResultSet.deleteRow
未能转换 java.sql.ResultSet.findColumn
未能转换 java.sql.ResultSet.first
未能转换 java.sql.ResultSet.getArray
未能转换 java.sql.ResultSet.getAsciiStream
未能转换 java.sql.ResultSet.getBigDecimal
未能转换 java.sql.ResultSet.getBinaryStream
未能转换 java.sql.ResultSet.getBlob
未能转换 java.sql.ResultSet.getBoolean
未能转换 java.sql.ResultSet.getByte
未能转换 java.sql.ResultSet.getBytes
未能转换 java.sql.ResultSet.getCharacterStream
未能转换 java.sql.ResultSet.getClob
未能转换 java.sql.ResultSet.getConcurrency
未能转换 java.sql.ResultSet.getCursorName
未能转换 java.sql.ResultSet.getDate
未能转换 java.sql.ResultSet.getDouble
未能转换 java.sql.ResultSet.getFetchDirection
未能转换 java.sql.ResultSet.getFetchSize
未能转换 java.sql.ResultSet.getFloat
未能转换 java.sql.ResultSet.getInt
未能转换 java.sql.ResultSet.getLong
未能转换 java.sql.ResultSet.getMetaData
未能转换 java.sql.ResultSet.getObject
未能转换 java.sql.ResultSet.getRef
未能转换 java.sql.ResultSet.getRow
未能转换 java.sql.ResultSet.getShort
未能转换 java.sql.ResultSet.getStatement
未能转换 java.sql.ResultSet.getString
未能转换 java.sql.ResultSet.getTime
未能转换 java.sql.ResultSet.getTimestamp
未能转换 java.sql.ResultSet.getType
未能转换 java.sql.ResultSet.getUnicodeStream
未能转换 java.sql.ResultSet.getWarnings
未能转换 java.sql.ResultSet.insertRow
未能转换 java.sql.ResultSet.isAfterLast
未能转换 java.sql.ResultSet.isBeforeFirst
未能转换 java.sql.ResultSet.isFirst
未能转换 java.sql.ResultSet.isLast
未能转换 java.sql.ResultSet.last

未能转换 java.sql.ResultSet.moveToCurrentRow
未能转换 java.sql.ResultSet.moveToInsertRow
未能转换 java.sql.ResultSet.next
未能转换 java.sql.ResultSet.previous
未能转换 java.sql.ResultSet.refreshRow
未能转换 java.sql.ResultSet.relative
未能转换 java.sql.ResultSet.rowDeleted
未能转换 java.sql.ResultSet.rowInserted
未能转换 java.sql.ResultSet.rowUpdated
未能转换 java.sql.ResultSet.setFetchDirection
未能转换 java.sql.ResultSet.setFetchSize
未能转换 java.sql.ResultSet.TYPE_FORWARD_ONLY
未能转换 java.sql.ResultSet.updateAsciiStream
未能转换 java.sql.ResultSet.updateBigDecimal
未能转换 java.sql.ResultSet.updateBinaryStream
未能转换 java.sql.ResultSet.updateBoolean
未能转换 java.sql.ResultSet.updateByte
未能转换 java.sql.ResultSet.updateBytes
未能转换 java.sql.ResultSet.updateCharacterStream
未能转换 java.sql.ResultSet.updateDate
未能转换 java.sql.ResultSet.updateDouble
未能转换 java.sql.ResultSet.updateFloat
未能转换 java.sql.ResultSet.updateInt
未能转换 java.sql.ResultSet.updateLong
未能转换 java.sql.ResultSet.updateNull
未能转换 java.sql.ResultSet.updateObject
未能转换 java.sql.ResultSet.updateRow
未能转换 java.sql.ResultSet.updateShort
未能转换 java.sql.ResultSet.updateString
未能转换 java.sql.ResultSet.updateTime
未能转换 java.sql.ResultSet.updateTimestamp
未能转换 java.sql.ResultSet.wasNull
未能转换 java.sql.ResultSetMetaData.columnNoNulls
未能转换 java.sql.ResultSetMetaData.columnNullable
未能转换 java.sql.ResultSetMetaData.columnNullableUnknown
未能转换 java.sql.ResultSetMetaData.getPrecision
未能转换 java.sql.ResultSetMetaData.isCaseSensitive
未能转换 java.sql.ResultSetMetaData.isCurrency
未能转换 java.sql.ResultSetMetaData.isDefinitelyWritable
未能转换 java.sql.ResultSetMetaData.isSearchable

未能转换 java.sql.ResultSetMetaData.isSigned
未能转换 java.sql.ResultSetMetaData.isWritable
未能转换 javax.sql.RowSetMetaData.setNullable
未能转换 java.sql.SQLData
未能转换 java.sql.SQLException.getNextException
未能转换 java.sql.SQLException.setNextException
未能转换 java.sql.SQLException.SQLException
未能转换 java.sql.SQLInput
未能转换 java.sql.SQLOutput
未能转换 java.sql.SQLPermission
未能转换 java.sql.SQLWarning
未能转换 java.sql.Statement.clearWarnings
未能转换 java.sql.Statement.close
未能转换 java.sql.Statement.execute
未能转换 java.sql.Statement.executeBatch
未能转换 java.sql.Statement.getFetchDirection
未能转换 java.sql.Statement.getFetchSize
未能转换 java.sql.Statement.getMaxFieldSize
未能转换 java.sql.Statement getMaxRows
未能转换 java.sql.Statement.getMoreResults
未能转换 java.sql.Statement.getResultSet
未能转换 java.sql.Statement.getResultSetConcurrency
未能转换 java.sql.Statement.getResultSetType
未能转换 java.sql.Statement.getUpdateCount
未能转换 java.sql.Statement.getWarnings
未能转换 java.sql.Statement.setCursorName
未能转换 java.sql.Statement.setEscapeProcessing
未能转换 java.sql.Statement.setFetchDirection
未能转换 java.sql.Statement.setFetchSize
未能转换 java.sql.Statement.setMaxFieldSize
未能转换 java.sql.Statement.setMaxRows
未能转换 java.sql.Struct
未能转换 java.sql.Timestamp.getNanos
未能转换 java.sql.Timestamp.setNanos
未能转换 java.sql.Timestamp.Timestamp
未能转换 java.sql.Types.<TypeName>

Java.text 错误信息

未能转换 java.text.Annotation
未能转换 java.text.AttributedCharacterIterator
未能转换 java.text.AttributedCharacterIterator.Attribute
未能转换 java.text.AttributedString
未能转换 java.text.BreakDictionary
未能转换 java.text.BreakIterator
未能转换 java.text.CharacterIterator.clone
未能转换 java.text.ChoiceFormat
未能转换 java.text.CollationElementIterator
未能转换 java.text.CollationKey.compareTo
未能转换 java.text.Collator.CANONICAL_DECOMPOSITION
未能转换 java.text.Collator.Collator
未能转换 java.text.Collator.compare
未能转换 java.text.Collator.FULL_DECOMPOSITION
未能转换 java.text.Collator.getDecomposition
未能转换 java.text.Collator.getStrength
未能转换 java.text.Collator.IDENTICAL
未能转换 java.text.Collator.NO_DECOMPOSITION
未能转换 java.text.Collator.PRIMARY
未能转换 java.text.Collator.SECONDARY
未能转换 java.text.Collator.setDecomposition
未能转换 java.text.Collator.setStrength
未能转换 java.text.Collator.TERTIARY
未能转换 java.text.DateFormat.format
未能转换 java.text.DateFormat.getNumberFormat
未能转换 java.text.DateFormat.getTimeZone
未能转换 java.text.DateFormat.isLenient
未能转换 java.text.DateFormat.numberFormat
未能转换 java.text.DateFormat.parse
未能转换 java.text.DateFormat.parseObject
未能转换 java.text.DateFormat.setLenient
未能转换 java.text.DateFormat.setNumberFormat
未能转换 java.text.DateFormat.setTimeZone
未能转换 java.text.DateFormatSymbols.DateFormatSymbols
未能转换 java.text.DateFormatSymbols.getEras
未能转换 java.text.DateFormatSymbols.getLocalPatternChars
未能转换 java.text.DateFormatSymbols.getZoneStrings

未能转换 java.text.DateFormatSymbols.setEras
未能转换 java.text.DateFormatSymbols.setLocalPatternChars
未能转换 java.text.DateFormatSymbols.setZoneStrings
未能转换 java.text.DecimalFormat
未能转换 java.text.DecimalFormatSymbols.DecimalFormatSymbols
未能转换 java.text.DecimalFormatSymbols.getDigit
未能转换 java.text.DecimalFormatSymbols.getInternationalCurrencySymbol
未能转换 java.text.DecimalFormatSymbols.getPatternSeparator
未能转换 java.text.DecimalFormatSymbols.getZeroDigit
未能转换 java.text.DecimalFormatSymbols.setDigit
未能转换 java.text.DecimalFormatSymbols.setInternationalCurrencySymbol
未能转换 java.text.DecimalFormatSymbols.setPatternSeparator
未能转换 java.text.DecimalFormatSymbols.setZeroDigit
未能转换 java.text.DictionaryBasedBreakIterator
未能转换 java.text.FieldPosition
未能转换 java.text.Format
未能转换 java.text.Format.format
未能转换 java.text.MessageFormat
未能转换 java.text.NumberFormat.format
未能转换 java.text.NumberFormat.FRACTION_FIELD
未能转换 java.text.NumberFormat.INTEGER_FIELD
未能转换 java.text.NumberFormat.isParseIntegerOnly
未能转换 java.text.NumberFormat.parse
未能转换 java.text.NumberFormat.parseObject
未能转换 java.text.NumberFormat.setParseIntegerOnly
未能转换 java.text.ParseException
未能转换 java.text.ParseException.getErrorOffset
未能转换 java.text.ParsePosition.getErrorIndex
未能转换 java.text.ParsePosition.setErrorIndex
未能转换 java.text.ParsePosition.toString
未能转换 java.text.RuleBasedBreakIterator
未能转换 java.text.RuleBasedCollator
未能转换 java.text.SimpleDateFormat
未能转换 java.text.StringCharacterIterator.clone
未能转换 java.text.StringCharacterIterator.setText

Java.text.resources 错误信息

未能转换 java.text.resources.BreakIteratorRules.BreakIteratorRules
未能转换 java.text.resources.BreakIteratorRules.getContents
未能转换 java.text.resources.DateFormatZoneData.DateFormatZoneData
未能转换 java.text.resources.DateFormatZoneData.getContents
未能转换 java.text.resources.DateFormatZoneData.getKeys
未能转换 java.text.resources.DateFormatZoneData_ar
未能转换 java.text.resources.DateFormatZoneData_ar.DateFormatZoneData_ar
未能转换 java.text.resources.DateFormatZoneData_ar.getContents
未能转换 java.text.resources.DateFormatZoneData_be
未能转换 java.text.resources.DateFormatZoneData_be.DateFormatZoneData_be
未能转换 java.text.resources.DateFormatZoneData_be.getContents
未能转换 java.text.resources.DateFormatZoneData_bg
未能转换 java.text.resources.DateFormatZoneData_bg.DateFormatZoneData_bg
未能转换 java.text.resources.DateFormatZoneData_bg.getContents
未能转换 java.text.resources.DateFormatZoneData_ca
未能转换 java.text.resources.DateFormatZoneData_ca.DateFormatZoneData_ca
未能转换 java.text.resources.DateFormatZoneData_ca.getContents
未能转换 java.text.resources.DateFormatZoneData_cs.DateFormatZoneData_cs
未能转换 java.text.resources.DateFormatZoneData_cs.getContents
未能转换 java.text.resources.DateFormatZoneData_da.DateFormatZoneData_da
未能转换 java.text.resources.DateFormatZoneData_da.getContents
未能转换 java.text.resources.DateFormatZoneData_de
未能转换 java.text.resources.DateFormatZoneData_de.DateFormatZoneData_de
未能转换 java.text.resources.DateFormatZoneData_de.getContents
未能转换 java.text.resources.DateFormatZoneData_de_AT
未能转换 java.text.resources.DateFormatZoneData_de_AT.DateFormatZoneData_de_AT
未能转换 java.text.resources.DateFormatZoneData_de_AT.getContents
未能转换 java.text.resources.DateFormatZoneData_de_CH
未能转换 java.text.resources.DateFormatZoneData_de_CH.DateFormatZoneData_de_CH
未能转换 java.text.resources.DateFormatZoneData_de_CH.getContents
未能转换 java.text.resources.DateFormatZoneData_el
未能转换 java.text.resources.DateFormatZoneData_el.DateFormatZoneData_el
未能转换 java.text.resources.DateFormatZoneData_el.getContents
未能转换 java.text.resources.DateFormatZoneData_en
未能转换 java.text.resources.DateFormatZoneData_en.DateFormatZoneData_en
未能转换 java.text.resources.DateFormatZoneData_en.getContents
未能转换 java.text.resources.DateFormatZoneData_en_CA

未能转换 java.text.resources.LocaleData.getKeys
未能转换 java.text.resources.LocaleData.init
未能转换 java.text.resources.LocaleData.LocaleData
未能转换 java.text.resources.LocaleElements.getContents
未能转换 java.text.resources.LocaleElements.LocaleElements
未能转换 java.text.resources.LocaleElements_ar
未能转换 java.text.resources.LocaleElements_ar.LocaleElements_ar
未能转换 java.text.resources.LocaleElements_ar_AE
未能转换 java.text.resources.LocaleElements_ar_AE.getContents
未能转换 java.text.resources.LocaleElements_ar_AE.LocaleElements_ar_AE
未能转换 java.text.resources.LocaleElements_ar_BH
未能转换 java.text.resources.LocaleElements_ar_BH.getContents
未能转换 java.text.resources.LocaleElements_ar_BH.LocaleElements_ar_BH
未能转换 java.text.resources.LocaleElements_ar_DZ
未能转换 java.text.resources.LocaleElements_ar_DZ.getContents
未能转换 java.text.resources.LocaleElements_ar_DZ.LocaleElements_ar_DZ
未能转换 java.text.resources.LocaleElements_ar_EG
未能转换 java.text.resources.LocaleElements_ar_EG.getContents
未能转换 java.text.resources.LocaleElements_ar_EG.LocaleElements_ar_EG
未能转换 java.text.resources.LocaleElements_ar_IQ
未能转换 java.text.resources.LocaleElements_ar_IQ.getContents
未能转换 java.text.resources.LocaleElements_ar_IQ.LocaleElements_ar_IQ
未能转换 java.text.resources.LocaleElements_ar_JO
未能转换 java.text.resources.LocaleElements_ar_JO.getContents
未能转换 java.text.resources.LocaleElements_ar_JO.LocaleElements_ar_JO
未能转换 java.text.resources.LocaleElements_ar_KW
未能转换 java.text.resources.LocaleElements_ar_KW.getContents
未能转换 java.text.resources.LocaleElements_ar_KW.LocaleElements_ar_KW
未能转换 java.text.resources.LocaleElements_ar_LB
未能转换 java.text.resources.LocaleElements_ar_LB.getContents
未能转换 java.text.resources.LocaleElements_ar_LB.LocaleElements_ar_LB
未能转换 java.text.resources.LocaleElements_ar LY.getContents
未能转换 java.text.resources.LocaleElements_ar LY.LocaleElements_ar LY
未能转换 java.text.resources.LocaleElements_ar_MA
未能转换 java.text.resources.LocaleElements_ar_MA.getContents
未能转换 java.text.resources.LocaleElements_ar_MA.LocaleElements_ar_MA
未能转换 java.text.resources.LocaleElements_ar_OM
未能转换 java.text.resources.LocaleElements_ar_OM.getContents
未能转换 java.text.resources.LocaleElements_ar_OM.LocaleElements_ar_OM
未能转换 java.text.resources.LocaleElements_ar_QA

未能转换 java.text.resources.LocaleElements_ar_QA.getContents
未能转换 java.text.resources.LocaleElements_ar_QA.LocaleElements_ar_QA
未能转换 java.text.resources.LocaleElements_ar_SA
未能转换 java.text.resources.LocaleElements_ar_SA.getContents
未能转换 java.text.resources.LocaleElements_ar_SA.LocaleElements_ar_SA
未能转换 java.text.resources.LocaleElements_ar_SD
未能转换 java.text.resources.LocaleElements_ar_SD.getContents
未能转换 java.text.resources.LocaleElements_ar_SD.LocaleElements_ar_SD
未能转换 java.text.resources.LocaleElements_ar_SY
未能转换 java.text.resources.LocaleElements_ar_SY.getContents
未能转换 java.text.resources.LocaleElements_ar_SY.LocaleElements_ar_SY
未能转换 java.text.resources.LocaleElements_ar_TN
未能转换 java.text.resources.LocaleElements_ar_TN.getContents
未能转换 java.text.resources.LocaleElements_ar_TN.LocaleElements_ar_TN
未能转换 java.text.resources.LocaleElements_ar_YE
未能转换 java.text.resources.LocaleElements_ar_YE.getContents
未能转换 java.text.resources.LocaleElements_ar_YE.LocaleElements_ar_YE
未能转换 java.text.resources.LocaleElements_be
未能转换 java.text.resources.LocaleElements_be.LocaleElements_be
未能转换 java.text.resources.LocaleElements_be_BY
未能转换 java.text.resources.LocaleElements_be_BY.getContents
未能转换 java.text.resources.LocaleElements_be_BY.LocaleElements_be_BY
未能转换 java.text.resources.LocaleElements_bg
未能转换 java.text.resources.LocaleElements_bg.LocaleElements_bg
未能转换 java.text.resources.LocaleElements_bg_BG
未能转换 java.text.resources.LocaleElements_bg_BG.getContents
未能转换 java.text.resources.LocaleElements_bg_BG.LocaleElements_bg_BG
未能转换 java.text.resources.LocaleElements_ca
未能转换 java.text.resources.LocaleElements_ca.LocaleElements_ca
未能转换 java.text.resources.LocaleElements_ca_ES
未能转换 java.text.resources.LocaleElements_ca_ES.getContents
未能转换 java.text.resources.LocaleElements_ca_ES.LocaleElements_ca_ES
未能转换 java.text.resources.LocaleElements_ca_ES_EURO
未能转换 java.text.resources.LocaleElements_ca_ES_EURO.getContents
未能转换 java.text.resources.LocaleElements_ca_ES_EURO.LocaleElements_ca_ES_EURO
未能转换 java.text.resources.LocaleElements_cs
未能转换 java.text.resources.LocaleElements_cs.LocaleElements_cs
未能转换 java.text.resources.LocaleElements_cs_CZ
未能转换 java.text.resources.LocaleElements_cs_CZ.getContents
未能转换 java.text.resources.LocaleElements_cs_CZ.LocaleElements_cs_CZ

未能转换 java.text.resources.LocaleElements_da
未能转换 java.text.resources.LocaleElements_da.LocaleElements_da
未能转换 java.text.resources.LocaleElements_da_DK
未能转换 java.text.resources.LocaleElements_da_DK.getContents
未能转换 java.text.resources.LocaleElements_da_DK.LocaleElements_da_DK
未能转换 java.text.resources.LocaleElements_de
未能转换 java.text.resources.LocaleElements_de.LocaleElements_de
未能转换 java.text.resources.LocaleElements_de_AT
未能转换 java.text.resources.LocaleElements_de_AT.LocaleElements_de_AT
未能转换 java.text.resources.LocaleElements_de_AT_EURO
未能转换 java.text.resources.LocaleElements_de_AT_EURO.getContents
未能转换 java.text.resources.LocaleElements_de_AT_EURO.LocaleElements_de_AT_EURO
未能转换 java.text.resources.LocaleElements_de_CH
未能转换 java.text.resources.LocaleElements_de_CH.LocaleElements_de_CH
未能转换 java.text.resources.LocaleElements_de_DE
未能转换 java.text.resources.LocaleElements_de_DE.getContents
未能转换 java.text.resources.LocaleElements_de_DE.LocaleElements_de_DE
未能转换 java.text.resources.LocaleElements_de_DE_EURO
未能转换 java.text.resources.LocaleElements_de_DE_EURO.getContents
未能转换 java.text.resources.LocaleElements_de_DE_EURO.LocaleElements_de_DE_EURO
未能转换 java.text.resources.LocaleElements_de_LU
未能转换 java.text.resources.LocaleElements_de_LU.getContents
未能转换 java.text.resources.LocaleElements_de_LU.LocaleElements_de_LU
未能转换 java.text.resources.LocaleElements_de_LU_EURO
未能转换 java.text.resources.LocaleElements_de_LU_EURO.getContents
未能转换 java.text.resources.LocaleElements_de_LU_EURO.LocaleElements_de_LU_EURO
未能转换 java.text.resources.LocaleElements_el
未能转换 java.text.resources.LocaleElements_el.LocaleElements_el
未能转换 java.text.resources.LocaleElements_el_GR
未能转换 java.text.resources.LocaleElements_el_GR.getContents
未能转换 java.text.resources.LocaleElements_el_GR.LocaleElements_el_GR
未能转换 java.text.resources.LocaleElements_en
未能转换 java.text.resources.LocaleElements_en.getContents
未能转换 java.text.resources.LocaleElements_en.LocaleElements_en
未能转换 java.text.resources.LocaleElements_en_AU
未能转换 java.text.resources.LocaleElements_en_AU.LocaleElements_en_AU
未能转换 java.text.resources.LocaleElements_en_CA
未能转换 java.text.resources.LocaleElements_en_CA.LocaleElements_en_CA
未能转换 java.text.resources.LocaleElements_en_GB
未能转换 java.text.resources.LocaleElements_en_GB.LocaleElements_en_GB

未能转换 java.text.resources.LocaleElements_fi
未能转换 java.text.resources.LocaleElements_fi.LocaleElements_fi
未能转换 java.text.resources.LocaleElements_fi_FI
未能转换 java.text.resources.LocaleElements_fi_FI.getContents
未能转换 java.text.resources.LocaleElements_fi_FI.LocaleElements_fi_FI
未能转换 java.text.resources.LocaleElements_fi_FI_EURO
未能转换 java.text.resources.LocaleElements_fi_FI_EURO.getContents
未能转换 java.text.resources.LocaleElements_fi_FI_EURO.LocaleElements_fi_FI_EURO
未能转换 java.text.resources.LocaleElements_fr
未能转换 java.text.resources.LocaleElements_fr.LocaleElements_fr
未能转换 java.text.resources.LocaleElements_fr_BE
未能转换 java.text.resources.LocaleElements_fr_BE.LocaleElements_fr_BE
未能转换 java.text.resources.LocaleElements_fr_BE_EURO
未能转换 java.text.resources.LocaleElements_fr_BE_EURO.getContents
未能转换 java.text.resources.LocaleElements_fr_BE_EURO.LocaleElements_fr_BE_EURO
未能转换 java.text.resources.LocaleElements_fr_CA
未能转换 java.text.resources.LocaleElements_fr_CA.LocaleElements_fr_CA
未能转换 java.text.resources.LocaleElements_fr_CH
未能转换 java.text.resources.LocaleElements_fr_CH.LocaleElements_fr_CH
未能转换 java.text.resources.LocaleElements_fr_FR
未能转换 java.text.resources.LocaleElements_fr_FR.getContents
未能转换 java.text.resources.LocaleElements_fr_FR.LocaleElements_fr_FR
未能转换 java.text.resources.LocaleElements_fr_FR_EURO
未能转换 java.text.resources.LocaleElements_fr_FR_EURO.getContents
未能转换 java.text.resources.LocaleElements_fr_FR_EURO.LocaleElements_fr_FR_EURO
未能转换 java.text.resources.LocaleElements_fr_LU
未能转换 java.text.resources.LocaleElements_fr_LU.getContents
未能转换 java.text.resources.LocaleElements_fr_LU.LocaleElements_fr_LU
未能转换 java.text.resources.LocaleElements_fr_LU_EURO
未能转换 java.text.resources.LocaleElements_fr_LU_EURO.getContents
未能转换 java.text.resources.LocaleElements_fr_LU_EURO.LocaleElements_fr_LU_EURO
未能转换 java.text.resources.LocaleElements_he
未能转换 java.text.resources.LocaleElements_he.LocaleElements_he
未能转换 java.text.resources.LocaleElements_hr
未能转换 java.text.resources.LocaleElements_hr.LocaleElements_hr
未能转换 java.text.resources.LocaleElements_hr_HR
未能转换 java.text.resources.LocaleElements_hr_HR.getContents
未能转换 java.text.resources.LocaleElements_hr_HR.LocaleElements_hr_HR
未能转换 java.text.resources.LocaleElements_hu
未能转换 java.text.resources.LocaleElements_hu.LocaleElements_hu

未能转换 java.text.resources.LocaleElements_hu_HU
未能转换 java.text.resources.LocaleElements_hu_HU.getContents
未能转换 java.text.resources.LocaleElements_hu_HU.LocaleElements_hu_HU
未能转换 java.text.resources.LocaleElements_is
未能转换 java.text.resources.LocaleElements_is.LocaleElements_is
未能转换 java.text.resources.LocaleElements_is_IS
未能转换 java.text.resources.LocaleElements_is_IS.getContents
未能转换 java.text.resources.LocaleElements_is_IS.LocaleElements_is_IS
未能转换 java.text.resources.LocaleElements_it
未能转换 java.text.resources.LocaleElements_it.LocaleElements_it
未能转换 java.text.resources.LocaleElements_it_CH
未能转换 java.text.resources.LocaleElements_it_CH.LocaleElements_it_CH
未能转换 java.text.resources.LocaleElements_it_IT
未能转换 java.text.resources.LocaleElements_it_IT.getContents
未能转换 java.text.resources.LocaleElements_it_IT.LocaleElements_it_IT
未能转换 java.text.resources.LocaleElements_it_IT_EURO
未能转换 java.text.resources.LocaleElements_it_IT_EURO.getContents
未能转换 java.text.resources.LocaleElements_it_IT_EURO.LocaleElements_it_IT_EURO
未能转换 java.text.resources.LocaleElements_iw
未能转换 java.text.resources.LocaleElements_iw.LocaleElements_iw
未能转换 java.text.resources.LocaleElements_iw_IL
未能转换 java.text.resources.LocaleElements_iw_IL.getContents
未能转换 java.text.resources.LocaleElements_iw_IL.LocaleElements_iw_IL
未能转换 java.text.resources.LocaleElements_ja
未能转换 java.text.resources.LocaleElements_ja.LocaleElements_ja
未能转换 java.text.resources.LocaleElements_ja_JP
未能转换 java.text.resources.LocaleElements_ja_JP.getContents
未能转换 java.text.resources.LocaleElements_ja_JP.LocaleElements_ja_JP
未能转换 java.text.resources.LocaleElements_ko
未能转换 java.text.resources.LocaleElements_ko.LocaleElements_ko
未能转换 java.text.resources.LocaleElements_ko_KR
未能转换 java.text.resources.LocaleElements_ko_KR.getContents
未能转换 java.text.resources.LocaleElements_ko_KR.LocaleElements_ko_KR
未能转换 java.text.resources.LocaleElements_lt
未能转换 java.text.resources.LocaleElements_lt.LocaleElements_lt
未能转换 java.text.resources.LocaleElements_lt_LT
未能转换 java.text.resources.LocaleElements_lt_LT.getContents
未能转换 java.text.resources.LocaleElements_lt_LT.LocaleElements_lt_LT
未能转换 java.text.resources.LocaleElements_lv
未能转换 java.text.resources.LocaleElements_lv.LocaleElements_lv

未能转换 java.text.resources.LocaleElements_lv_LV
未能转换 java.text.resources.LocaleElements_lv_LV.getContents
未能转换 java.text.resources.LocaleElements_lv_LV.LocaleElements_lv_LV
未能转换 java.text.resources.LocaleElements_mk
未能转换 java.text.resources.LocaleElements_mk.LocaleElements_mk
未能转换 java.text.resources.LocaleElements_mk_MK
未能转换 java.text.resources.LocaleElements_mk_MK.getContents
未能转换 java.text.resources.LocaleElements_mk_MK.LocaleElements_mk_MK
未能转换 java.text.resources.LocaleElements_nl
未能转换 java.text.resources.LocaleElements_nl.LocaleElements_nl
未能转换 java.text.resources.LocaleElements_nl_BE
未能转换 java.text.resources.LocaleElements_nl_BE.LocaleElements_nl_BE
未能转换 java.text.resources.LocaleElements_nl_BE_EURO
未能转换 java.text.resources.LocaleElements_nl_BE_EURO.getContents
未能转换 java.text.resources.LocaleElements_nl_BE_EURO.LocaleElements_nl_BE_EURO
未能转换 java.text.resources.LocaleElements_nl_NL
未能转换 java.text.resources.LocaleElements_nl_NL.getContents
未能转换 java.text.resources.LocaleElements_nl_NL.LocaleElements_nl_NL
未能转换 java.text.resources.LocaleElements_nl_NL_EURO
未能转换 java.text.resources.LocaleElements_nl_NL_EURO.getContents
未能转换 java.text.resources.LocaleElements_nl_NL_EURO.LocaleElements_nl_NL_EURO
未能转换 java.text.resources.LocaleElements_no
未能转换 java.text.resources.LocaleElements_no.LocaleElements_no
未能转换 java.text.resources.LocaleElements_no_NO
未能转换 java.text.resources.LocaleElements_no_NO.getContents
未能转换 java.text.resources.LocaleElements_no_NO.LocaleElements_no_NO
未能转换 java.text.resources.LocaleElements_no_NO_NY
未能转换 java.text.resources.LocaleElements_no_NO_NY.LocaleElements_no_NO_NY
未能转换 java.text.resources.LocaleElements_pl
未能转换 java.text.resources.LocaleElements_pl.LocaleElements_pl
未能转换 java.text.resources.LocaleElements_pl_PL
未能转换 java.text.resources.LocaleElements_pl_PL.getContents
未能转换 java.text.resources.LocaleElements_pl_PL.LocaleElements_pl_PL
未能转换 java.text.resources.LocaleElements_pt
未能转换 java.text.resources.LocaleElements_pt.LocaleElements_pt
未能转换 java.text.resources.LocaleElements_pt_BR
未能转换 java.text.resources.LocaleElements_pt_BR.getContents
未能转换 java.text.resources.LocaleElements_pt_BR.LocaleElements_pt_BR
未能转换 java.text.resources.LocaleElements_pt_BZ
未能转换 java.text.resources.LocaleElements_pt_BZ.LocaleElements_pt_BZ

未能转换 java.text.resources.LocaleElements_pt_PT
未能转换 java.text.resources.LocaleElements_pt_PT.getContents
未能转换 java.text.resources.LocaleElements_pt_PT.LocaleElements_pt_PT
未能转换 java.text.resources.LocaleElements_pt_PT_EURO
未能转换 java.text.resources.LocaleElements_pt_PT_EURO.getContents
未能转换 java.text.resources.LocaleElements_pt_PT_EURO.LocaleElements_pt_PT_EURO
未能转换 java.text.resources.LocaleElements_ro
未能转换 java.text.resources.LocaleElements_ro.LocaleElements_ro
未能转换 java.text.resources.LocaleElements_ro_RO
未能转换 java.text.resources.LocaleElements_ro_RO.getContents
未能转换 java.text.resources.LocaleElements_ro_RO.LocaleElements_ro_RO
未能转换 java.text.resources.LocaleElements_ru
未能转换 java.text.resources.LocaleElements_ru.LocaleElements_ru
未能转换 java.text.resources.LocaleElements_ru_RU
未能转换 java.text.resources.LocaleElements_ru_RU.getContents
未能转换 java.text.resources.LocaleElements_ru_RU.LocaleElements_ru_RU
未能转换 java.text.resources.LocaleElements_sh
未能转换 java.text.resources.LocaleElements_sh.LocaleElements_sh
未能转换 java.text.resources.LocaleElements_sh_YU
未能转换 java.text.resources.LocaleElements_sh_YU.getContents
未能转换 java.text.resources.LocaleElements_sh_YU.LocaleElements_sh_YU
未能转换 java.text.resources.LocaleElements_sk
未能转换 java.text.resources.LocaleElements_sk.LocaleElements_sk
未能转换 java.text.resources.LocaleElements_sk_SK
未能转换 java.text.resources.LocaleElements_sk_SK.getContents
未能转换 java.text.resources.LocaleElements_sk_SK.LocaleElements_sk_SK
未能转换 java.text.resources.LocaleElements_sl
未能转换 java.text.resources.LocaleElements_sl.LocaleElements_sl
未能转换 java.text.resources.LocaleElements_sl_SI
未能转换 java.text.resources.LocaleElements_sl_SI.getContents
未能转换 java.text.resources.LocaleElements_sl_SI.LocaleElements_sl_SI
未能转换 java.text.resources.LocaleElements_sq
未能转换 java.text.resources.LocaleElements_sq.LocaleElements_sq
未能转换 java.text.resources.LocaleElements_sq_AL
未能转换 java.text.resources.LocaleElements_sq_AL.getContents
未能转换 java.text.resources.LocaleElements_sq_AL.LocaleElements_sq_AL
未能转换 java.text.resources.LocaleElements_sr
未能转换 java.text.resources.LocaleElements_sr.LocaleElements_sr
未能转换 java.text.resources.LocaleElements_sr_YU
未能转换 java.text.resources.LocaleElements_sr_YU.getContents

未能转换 java.text.resources.LocaleElements_sr_YU.LocaleElements_sr_YU
未能转换 java.text.resources.LocaleElements_sv
未能转换 java.text.resources.LocaleElements_sv.LocaleElements_sv
未能转换 java.text.resources.LocaleElements_sv_SE
未能转换 java.text.resources.LocaleElements_sv_SE.getContents
未能转换 java.text.resources.LocaleElements_sv_SE.LocaleElements_sv_SE
未能转换 java.text.resources.LocaleElements_th
未能转换 java.text.resources.LocaleElements_th.getContents
未能转换 java.text.resources.LocaleElements_th.LocaleElements_th
未能转换 java.text.resources.LocaleElements_th_TH
未能转换 java.text.resources.LocaleElements_th_TH.getContents
未能转换 java.text.resources.LocaleElements_th_TH.LocaleElements_th_TH
未能转换 java.text.resources.LocaleElements_tr
未能转换 java.text.resources.LocaleElements_tr.LocaleElements_tr
未能转换 java.text.resources.LocaleElements_tr_TR
未能转换 java.text.resources.LocaleElements_tr_TR.getContents
未能转换 java.text.resources.LocaleElements_tr_TR.LocaleElements_tr_TR
未能转换 java.text.resources.LocaleElements_uk
未能转换 java.text.resources.LocaleElements_uk.LocaleElements_uk
未能转换 java.text.resources.LocaleElements_uk_UA
未能转换 java.text.resources.LocaleElements_uk_UA.getContents
未能转换 java.text.resources.LocaleElements_uk_UA.LocaleElements_uk_UA
未能转换 java.text.resources.LocaleElements_zh
未能转换 java.text.resources.LocaleElements_zh.LocaleElements_zh
未能转换 java.text.resources.LocaleElements_zh_CN
未能转换 java.text.resources.LocaleElements_zh_CN.getContents
未能转换 java.text.resources.LocaleElements_zh_CN.LocaleElements_zh_CN
未能转换 java.text.resources.LocaleElements_zh_HK
未能转换 java.text.resources.LocaleElements_zh_HK.getContents
未能转换 java.text.resources.LocaleElements_zh_HK.LocaleElements_zh_HK
未能转换 java.text.resources.LocaleElements_zh_SG
未能转换 java.text.resources.LocaleElements_zh_SG.LocaleElements_zh_SG
未能转换 java.text.resources.LocaleElements_zh_TW
未能转换 java.text.resources.LocaleElements_zh_TW.LocaleElements_zh_TW

Java.util 错误信息

未能转换 java.util.AbstractCollection.AbstractCollection
未能转换 java.util.AbstractCollection.add
未能转换 java.util.AbstractCollection.addAll
未能转换 java.util.AbstractCollection.contains
未能转换 java.util.AbstractCollection.containsAll
未能转换 java.util.AbstractCollection.iterator
未能转换 java.util.AbstractCollection.remove
未能转换 java.util.AbstractCollection.removeAll
未能转换 java.util.AbstractCollection.retainAll
未能转换 java.util.AbstractCollection.toArray
未能转换 java.util.AbstractCollection.toArray(Object[])
未能转换 java.util.AbstractList.AbstractList
未能转换 java.util.AbstractList.add
未能转换 java.util.AbstractList.addAll
未能转换 java.util.AbstractList.lastIndexOf
未能转换 java.util.AbstractList.listIterator
未能转换 java.util.AbstractList.listIterator(int)
未能转换 java.util.AbstractList.modCount
未能转换 java.util.AbstractList.remove
未能转换 java.util.AbstractList.removeRange
未能转换 java.util.AbstractList.subList
未能转换 java.util.AbstractMap
未能转换 java.util.AbstractMap.AbstractMap
未能转换 java.util.AbstractMap.clear
未能转换 java.util.AbstractMap.containsKey
未能转换 java.util.AbstractMap.containsValue
未能转换 java.util.AbstractMap.entrySet
未能转换 java.util.AbstractMap.equals
未能转换 java.util.AbstractMap.get
未能转换 java.util.AbstractMap.hashCode
未能转换 java.util.AbstractMap.keySet
未能转换 java.util.AbstractMap.put
未能转换 java.util.AbstractMap.putAll
未能转换 java.util.AbstractMap.remove
未能转换 java.util.AbstractMap.size
未能转换 java.util.AbstractMap.values
未能转换 java.util.AbstractSequentialList.AbstractSequentialList

未能转换 java.util.AbstractSequentialList.addAll
未能转换 java.util.AbstractSequentialList.listIterator
未能转换 java.util.AbstractSequentialList.remove
未能转换 java.util.AbstractSequentialList.set
未能转换 java.util.AbstractSet.AbstractSet
未能转换 java.util.AbstractSet.removeAll
未能转换 java.util.ArrayList.add
未能转换 java.util.ArrayList.toArray
未能转换 java.util.Arrays.asList
未能转换 java.util.Arrays.sort
未能转换 java.util.BitSet.andNot
未能转换 java.util.BitSet.equals
未能转换 java.util.BitSet.length
未能转换 java.util.Calendar.add
未能转换 java.util.Calendar.after
未能转换 java.util.Calendar.AM
未能转换 java.util.Calendar.AM_PM
未能转换 java.util.Calendar.APRIL
未能转换 java.util.Calendar.areFieldsSet
未能转换 java.util.Calendar.AUGUST
未能转换 java.util.Calendar.before
未能转换 java.util.Calendar.Calendar
未能转换 java.util.Calendar.clear
未能转换 java.util.Calendar.clone
未能转换 java.util.Calendar.complete
未能转换 java.util.Calendar.computeFields
未能转换 java.util.Calendar.computeTime
未能转换 java.util.Calendar.DAY_OF_MONTH
未能转换 java.util.Calendar.DAY_OF_WEEK
未能转换 java.util.Calendar.DAY_OF_WEEK_IN_MONTH
未能转换 java.util.Calendar.DAY_OF_YEAR
未能转换 java.util.Calendar.DECEMBER
未能转换 java.util.Calendar.DST_OFFSET
未能转换 java.util.Calendar.ERA
未能转换 java.util.Calendar.FEBRUARY
未能转换 java.util.Calendar.FIELD_COUNT
未能转换 java.util.Calendar.fields
未能转换 java.util.Calendar.get
未能转换 java.util.Calendar.getActualMaximum
未能转换 java.util.Calendar.getActualMinimum

未能转换 java.util.Calendar.getAvailableLocales
未能转换 java.util.Calendar.getGreatestMinimum
未能转换 java.util.Calendar.getInstance
未能转换 java.util.Calendar.getLeastMaximum
未能转换 java.util.Calendar.getMaximum
未能转换 java.util.Calendar.getMinimalDaysInFirstWeek
未能转换 java.util.Calendar.getMinimum
未能转换 java.util.Calendar.getTimeInMillis
未能转换 java.util.Calendar.getTimeZone
未能转换 java.util.Calendar.HOUR
未能转换 java.util.Calendar.HOUR_OF_DAY
未能转换 java.util.Calendar.internalGet
未能转换 java.util.Calendar.isLenient
未能转换 java.util.Calendar.isSet
未能转换 java.util.Calendar.isSet~
未能转换 java.util.Calendar.isTimeSet
未能转换 java.util.Calendar.JANUARY
未能转换 java.util.Calendar.JULY
未能转换 java.util.Calendar.JUNE
未能转换 java.util.Calendar.MARCH
未能转换 java.util.Calendar.MAY
未能转换 java.util.Calendar.NOVEMBER
未能转换 java.util.Calendar.OCTOBER
未能转换 java.util.Calendar.PM
未能转换 java.util.Calendar.roll
未能转换 java.util.Calendar.roll(int, boolean)
未能转换 java.util.Calendar.roll(int, int)
未能转换 java.util.Calendar.SEPTEMBER
未能转换 java.util.Calendar.setLenient
未能转换 java.util.Calendar.setMinimalDaysInFirstWeek
未能转换 java.util.Calendar.setTimeInMillis
未能转换 java.util.Calendar.setTimeZone
未能转换 java.util.Calendar.time
未能转换 java.util.Calendar.UNDECIMBER
未能转换 java.util.Calendar.WEEK_OF_MONTH
未能转换 java.util.Calendar.WEEK_OF_YEAR
未能转换 java.util.Calendar.ZONE_OFFSET
未能转换 java.util.Collection.add
未能转换 java.util.Collection.addAll
未能转换 java.util.Collection.clear

未能转换 java.util.Collection.contains
未能转换 java.util.Collection.containsAll
未能转换 java.util.Collection.remove
未能转换 java.util.Collection.removeAll
未能转换 java.util.Collection.retainAll
未能转换 java.util.Collection.toArray
未能转换 java.util.Collections
未能转换 java.util.Collections.EMPTY_MAP
未能转换 java.util.Collections.EMPTY_SET
未能转换 java.util.Collections.max(Collection)
未能转换 java.util.Collections.max(Collection, Comparator)
未能转换 java.util.Collections.min(Collection)
未能转换 java.util.Collections.min(Collection, Comparator)
未能转换 java.util.Collections.reverseOrder
未能转换 java.util.Collections.sort(Comparator)
未能转换 java.util.Collections.sort(List)
未能转换 java.util.Collections.synchronizedMap
未能转换 java.util.Collections.synchronizedSortedSet
未能转换 java.util.Collections.unmodifiableMap
未能转换 java.util.Collections.unmodifiableSortedMap
未能转换 java.util.Collections.unmodifiableSortedSet
未能转换 java.util.Date.clone
未能转换 java.util.Date.Date(int, int, int)
未能转换 java.util.Date.Date(int, int, int, int, int)
未能转换 java.util.Date.Date(int, int, int, int, int, int)
未能转换 java.util.Date.Date(long)
未能转换 java.util.Date.Date(String)
未能转换 java.util.Date.getMonth
未能转换 java.util.Date.getTime
未能转换 java.util.Date.getTimezoneOffset
未能转换 java.util.Date.getYear
未能转换 java.util.Date.hashCode
未能转换 java.util.Date.parse
未能转换 java.util.Date.setTime
未能转换 java.util.Date.toGMTString
未能转换 java.util.Date.toLocaleString
未能转换 java.util.Date.toString
未能转换 java.util.Date.UTC
未能转换 java.util.Dictionary.Dictionary
未能转换 java.util.Dictionary.get

未能转换 java.util.Dictionary.isEmpty
未能转换 java.util.Dictionary.keys
未能转换 java.util.Dictionary.put
未能转换 java.util.Dictionary.remove
未能转换 java.util.Enumeration.hasMoreElements
未能转换 java.util.Enumeration.nextElement
未能转换 java.util.Entry.setValue
未能转换 java.util.EventListener
未能转换 java.util.EventObject
未能转换 java.util.EventObject.source
未能转换 java.util.GregorianCalendar.add
未能转换 java.util.GregorianCalendar.after
未能转换 java.util.GregorianCalendar.before
未能转换 java.util.GregorianCalendar.computeFields
未能转换 java.util.GregorianCalendar.computeTime
未能转换 java.util.GregorianCalendar.getActualMaximum
未能转换 java.util.GregorianCalendar.getActualMinimum
未能转换 java.util.GregorianCalendar.getGreatestMinimum
未能转换 java.util.GregorianCalendar.getGregorianChange
未能转换 java.util.GregorianCalendar.getLeastMaximum
未能转换 java.util.GregorianCalendar.getMaximum
未能转换 java.util.GregorianCalendar.getMinimum
未能转换 java.util.GregorianCalendar.GregorianCalendar
未能转换 java.util.GregorianCalendar.roll
未能转换 java.util.GregorianCalendar.roll(int, boolean)
未能转换 java.util.GregorianCalendar.setGregorianChange
未能转换 java.util.HashMap
未能转换 java.util.HashMap.entrySet
未能转换 java.util.HashMap.get
未能转换 java.util.HashMap.keySet
未能转换 java.util.HashMap.putAll
未能转换 java.util.HashSet
未能转换 java.util.HashSet.add
未能转换 java.util.HashSet.clear
未能转换 java.util.HashSet.clone
未能转换 java.util.HashSet.contains
未能转换 java.util.HashSet.HashSet
未能转换 java.util.HashSet.HashSet(Collection)
未能转换 java.util.HashSet.HashSet(int)
未能转换 java.util.HashSet.HashSet(int, float)

未能转换 java.util.HashSet.remove
未能转换 java.util.Hashtable.entrySet
未能转换 java.util.Hashtable.putAll
未能转换 java.util.Hashtable.rehash
未能转换 java.util.HashtableEntry
未能转换 java.util.HashtableEnumerator
未能转换 java.util.Iterator.hasNext
未能转换 java.util.Iterator.next
未能转换 java.util.Iterator.remove
未能转换 java.util.LinkedList
未能转换 java.util.LinkedList.add
未能转换 java.util.LinkedList.listIterator
未能转换 java.util.List.add
未能转换 java.util.List.addAll
未能转换 java.util.List.containsAll
未能转换 java.util.List.lastIndexOf
未能转换 java.util.List.listIterator
未能转换 java.util.List.remove
未能转换 java.util.List.removeAll
未能转换 java.util.List retainAll
未能转换 java.util.List subList
未能转换 java.util.List toArray
未能转换 java.util.ListIterator.add
未能转换 java.util.ListIterator.hasNext
未能转换 java.util.ListIterator.hasPrevious
未能转换 java.util.ListIterator.next
未能转换 java.util.ListIterator.nextInt
未能转换 java.util.ListIterator.previous
未能转换 java.util.ListIterator.previousIndex
未能转换 java.util.ListIterator.remove
未能转换 java.util.ListIterator.set
未能转换 java.util.ListResourceBundle.getContents
未能转换 java.util.ListResourceBundle.getKeys
未能转换 java.util.ListResourceBundle.handleGetObject
未能转换 java.util.ListResourceBundle.ListResourceBundle
未能转换 java.util.Locale.CHINESE
未能转换 java.util.Locale.getDefault
未能转换 java.util.Locale.getDisplayName
未能转换 java.util.Locale.getDisplayVariant
未能转换 java.util.Locale.getISOCountries

未能转换 java.util.Locale.getLanguage
未能转换 java.util.Locale.getVariant
未能转换 java.util.Locale.hashCode
未能转换 java.util.Locale.setDefault
未能转换 java.util.Map.containsValue
未能转换 java.util.Map.entrySet
未能转换 java.util.Map.get
未能转换 java.util.Map.isEmpty
未能转换 java.util.Map.keySet
未能转换 java.util.Map.putAll
未能转换 java.util.MissingResourceException.getClassName
未能转换 java.util.MissingResourceException.getKey
未能转换 java.util.MissingResourceException.MissingResourceException
未能转换 java.util.Properties.defaults
未能转换 java.util.Properties.list
未能转换 java.util.Properties.load
未能转换 java.util.Properties.save
未能转换 java.util.Properties.setProperty
未能转换 java.util.Properties.store
未能转换 java.util.PropertyPermission
未能转换 java.util.PropertyResourceBundle.getKeys
未能转换 java.util.PropertyResourceBundle.PropertyResourceBundle
未能转换 java.util.Random.nextLong
未能转换 java.util.Random.nextBoolean
未能转换 java.util.Random.nextGaussian
未能转换 java.util.ResourceBundle.getKeys
未能转换 java.util.ResourceBundle.getObject
未能转换 java.util.ResourceBundle.getString
未能转换 java.util.ResourceBundle.getStringArray
未能转换 java.util.ResourceBundle.handleGetObject
未能转换 java.util.ResourceBundle.parent
未能转换 java.util.ResourceBundle.ResourceBundle
未能转换 java.util.ResourceBundle.setParent
未能转换 java.util.Set.add
未能转换 java.util.Set.addAll
未能转换 java.util.Set.clear
未能转换 java.util.Set.contains
未能转换 java.util.Set.containsAll
未能转换 java.util.Set.remove
未能转换 java.util.Set.removeAll

未能转换 java.util.Set retainAll
未能转换 java.util.Set toArray
未能转换 java.util.Set toArray(Object[])
未能转换 java.util.SimpleTimeZone
未能转换 java.util.SimpleTimeZone clone
未能转换 java.util.SimpleTimeZone getRawOffset
未能转换 java.util.SimpleTimeZone setEndRule
未能转换 java.util.SimpleTimeZone setRawOffset
未能转换 java.util.SimpleTimeZone setStartRule
未能转换 java.util.SimpleTimeZone setStartYear
未能转换 java.util.SimpleTimeZone SimpleTimeZone
未能转换 java.util.SortedMap
未能转换 java.util.SortedMap comparator
未能转换 java.util.SortedMap headMap
未能转换 java.util.SortedMap subMap
未能转换 java.util.SortedMap tailMap
未能转换 java.util.SortedSet comparator
未能转换 java.util.Stack addElement
未能转换 java.util.Stack elements
未能转换 java.util.StringTokenizer StringTokenizer
未能转换 java.util.SystemClassLoader
未能转换 java.util.Timer cancel
未能转换 java.util.Timer schedule(TimerTask, Date)
未能转换 java.util.Timer schedule(TimerTask, Date, long)
未能转换 java.util.Timer schedule(TimerTask, long)
未能转换 java.util.Timer schedule(TimerTask, long, long)
未能转换 java.util.Timer scheduleAtFixedRate(TimerTask, Date, long)
未能转换 java.util.Timer scheduleAtFixedRate(TimerTask, long, long)
未能转换 java.util.Timer Timer
未能转换 java.util.TimerTask
未能转换 java.util.TimeZone clone
未能转换 java.util.TimeZone getAvailableIDs
未能转换 java.util.TimeZone getDisplayName(boolean, int)
未能转换 java.util.TimeZone getDisplayName(boolean, int, Locale)
未能转换 java.util.TimeZone getDisplayName(Locale)
未能转换 java.util.TimeZone getRawOffset
未能转换 java.util.TimeZone getTimeZone
未能转换 java.util.TimeZone hasSameRules
未能转换 java.util.TimeZone setDefault
未能转换 java.util.TimeZone setID

未能转换 java.util.TimeZone.setRawOffset
未能转换 java.util.TimeZone.TimeZone
未能转换 java.util.TreeMap.comparator
未能转换 java.util.TreeMap.entrySet
未能转换 java.util.TreeMap.headMap
未能转换 java.util.TreeMap.keySet
未能转换 java.util.TreeMap.putAll
未能转换 java.util.TreeMap.subMap
未能转换 java.util.TreeMap.tailMap
未能转换 java.util.TreeMap.TreeMap
未能转换 java.util.TreeMap.TreeMap(Map)
未能转换 java.util.TreeMap.TreeMap(SortedMap)
未能转换 java.util.TreeMap.values
未能转换 java.util.TreeSet
未能转换 java.util.TreeSet.comparator
未能转换 java.util.TreeSet.TreeSet(Comparator)
未能转换 java.util.TreeSet.TreeSet(SortedMap)
未能转换 java.util.TreeSet.TreeSet(SortedSet)
未能转换 java.util.Vector.add
未能转换 java.util.Vector.capacityIncrement
未能转换 java.util.Vector.containsAll
未能转换 java.util.Vector.elementAt
未能转换 java.util.Vector.removeAll
未能转换 java.util.Vector.retainAll
未能转换 java.util.Vector.toArray
未能转换 java.util.Vector.Vector
未能转换 java.util.VectorEnumerator
未能转换 java.util.WeakHashMap
未能转换 java.util.WeakHashMap.entrySet
未能转换 java.util.WeakHashMap.WeakHashMap

Java.util.cab 错误信息

未能转换 com.ms.util.cab.CabConstants

未能转换 com.ms.util.cab.CabCorruptException

未能转换 com.ms.util.cab.CabCreator

未能转换 com.ms.util.cab.CabDecoder

未能转换 com.ms.util.cab.CabDecoderInterface

Java.util.jar 错误信息

未能转换 java.util.jar.Attributes

未能转换 java.util.jar.Attributes.Name

未能转换 java.util.jar.JarEntry

未能转换 java.util.jar.JarException

未能转换 java.util.jar.JarFile

未能转换 java.util.jar.JarInputStream

未能转换 java.util.jar.JarOutputStream

未能转换 java.util.jar.Manifest

Java.util.zip 错误信息

未能转换 java.util.zip.Adler32
未能转换 java.util.zip.CheckedInputStream
未能转换 java.util.zip.CheckedInputStream.CheckedInputStream
未能转换 java.util.zip.CheckedInputStream.getChecksum
未能转换 java.util.zip.CheckedOutputStream
未能转换 java.util.zip.CheckedOutputStream.CheckedOutputStream
未能转换 java.util.zip.CheckedOutputStream.getChecksum
未能转换 java.util.zip.Checksum
未能转换 java.util.zip.CRC32
未能转换 java.util.zip.DataFormatException
未能转换 java.util.zip.Deflater
未能转换 java.util.zip.DeflaterOutputStream
未能转换 java.util.zip.GZIPInputStream
未能转换 java.util.zip.GZIPOutputStream
未能转换 java.util.zip.Inflater
未能转换 java.util.zip.InflaterInputStream
未能转换 java.util.zip.ZipEntry
未能转换 java.util.zip.ZipFile
未能转换 java.util.zip.ZipInputStream
未能转换 java.util.zip.ZipOutputStream

Javax.accessibility 错误信息

未能转换 javax.accessibility.AccessibleAction
未能转换 javax.accessibility.AccessibleAction.doAccessibleAction
未能转换 javax.accessibility.AccessibleAction.getAccessibleActionCount
未能转换 javax.accessibility.AccessibleAction.getAccessibleActionDescription
未能转换 javax.accessibility.AccessibleBundle
未能转换 javax.accessibility.AccessibleComponent
未能转换 javax.accessibility.AccessibleComponent.addFocusListener
未能转换 javax.accessibility.AccessibleComponent.isVisible
未能转换 javax.accessibility.AccessibleComponent.removeFocusListener
未能转换 javax.accessibility.AccessibleComponent.setBounds
未能转换 javax.accessibility.AccessibleContext.<PropertyChangeEvent>
未能转换 javax.accessibility.AccessibleContext.accessibleDescription
未能转换 javax.accessibility.AccessibleContext.accessibleName
未能转换 javax.accessibility.AccessibleContext.accessibleParent
未能转换 javax.accessibility.AccessibleContext.addPropertyChangeListener
未能转换 javax.accessibility.AccessibleContext.firePropertyChange
未能转换 javax.accessibility.AccessibleContext.getAccessibleIcon
未能转换 javax.accessibility.AccessibleContext.getAccessibleIndexInParent
未能转换 javax.accessibility.AccessibleContext.getAccessibleParent
未能转换 javax.accessibility.AccessibleContext.getAccessibleRelationSet
未能转换 javax.accessibility.AccessibleContext.getAccessibleSelection
未能转换 javax.accessibility.AccessibleContext.getAccessibleTable
未能转换 javax.accessibility.AccessibleContext.getAccessibleText
未能转换 javax.accessibility.AccessibleContext.removePropertyChangeListener
未能转换 javax.accessibility.AccessibleContext.setAccessibleParent
未能转换 javax.accessibility.AccessibleHyperlink
未能转换 javax.accessibility.AccessibleHypertext
未能转换 javax.accessibility.AccessibleIcon
未能转换 javax.accessibility.AccessibleRelation
未能转换 javax.accessibility.AccessibleRelationSet
未能转换 javax.accessibility.AccessibleResourceBundle
未能转换 javax.accessibility.AccessibleRole.AccessibleRole
未能转换 javax.accessibility.AccessibleSelection
未能转换 javax.accessibility.AccessibleSelection.addAccessibleSelection
未能转换 javax.accessibility.AccessibleSelection.clearAccessibleSelection
未能转换 javax.accessibility.AccessibleSelection.getAccessibleSelection
未能转换 javax.accessibility.AccessibleSelection.getAccessibleSelectionCount

未能转换 javax.accessibility.AccessibleSelection.isAccessibleChildSelected
未能转换 javax.accessibility.AccessibleSelection.removeAccessibleSelection
未能转换 javax.accessibility.AccessibleSelection.selectAllAccessibleSelection
未能转换 javax.accessibility.AccessibleState.<StateName>
未能转换 javax.accessibility.AccessibleState.AccessibleState
未能转换 javax.accessibility.AccessibleStateSet.states
未能转换 javax.accessibility.AccessibleStateSet.toArray
未能转换 javax.accessibility.AccessibleTable
未能转换 javax.accessibility.AccessibleTableModelChange
未能转换 javax.accessibility.AccessibleText
未能转换 javax.accessibility.AccessibleText.<TextType>
未能转换 javax.accessibility.AccessibleText.getAfterIndex
未能转换 javax.accessibility.AccessibleText.getAtIndex
未能转换 javax.accessibility.AccessibleText.getBeforeIndex
未能转换 javax.accessibility.AccessibleText.getCaretPosition
未能转换 javax.accessibility.AccessibleText.getCharacterAttribute
未能转换 javax.accessibility.AccessibleText.getCharacterBounds
未能转换 javax.accessibility.AccessibleText.getCharCount
未能转换 javax.accessibility.AccessibleText.getIndexAtPoint
未能转换 javax.accessibility.AccessibleText.getSelectedText
未能转换 javax.accessibility.AccessibleText.getSelectionEnd
未能转换 javax.accessibility.AccessibleText.getSelectionStart
未能转换 javax.accessibility.AccessibleValue
未能转换 javax.accessibility.AccessibleValue.getCurrentAccessibleValue
未能转换 javax.accessibility.AccessibleValue.getMaximumAccessibleValue
未能转换 javax.accessibility.AccessibleValue.getMinimumAccessibleValue
未能转换 javax.accessibility.AccessibleValue.setCurrentAccessibleValue

Java.crypto 错误信息

未能转换 javax.crypto.Cipher.<KeyType>
未能转换 javax.crypto.Cipher.Cipher
未能转换 javax.crypto.Cipher.doFinal
未能转换 javax.crypto.Cipher.getExemptionMechanism
未能转换 javax.crypto.Cipher.getInstance
未能转换 javax.crypto.Cipher.getParameters
未能转换 javax.crypto.Cipher.getProvider
未能转换 javax.crypto.Cipher.init
未能转换 javax.crypto.Cipher.t
未能转换 javax.crypto.Cipher.unwrap
未能转换 javax.crypto.Cipher.UNWRAP_MODE
未能转换 javax.crypto.Cipher.wrap
未能转换 javax.crypto.Cipher.WRAP_MODE
未能转换 javax.crypto.CipherInputStream.available
未能转换 javax.crypto.CipherInputStream.CipherInputStream(Stream)
未能转换 javax.crypto.CipherInputStream.CipherInputStream(Stream, Cipher)
未能转换 javax.crypto.CipherInputStream.skip
未能转换 javax.crypto.CipherOutputStream.CipherOutputStream(OutputStream)
未能转换 javax.crypto.CipherOutputStream.CipherOutputStream(OutputStream, Cipher)
未能转换 javax.crypto.CipherSpi
未能转换 javax.crypto.ExemptionMechanism
未能转换 javax.crypto.ExemptionMechanismSpi
未能转换 javax.crypto.interfaces.DHKey
未能转换 javax.crypto.interfaces.DHPrivateKey
未能转换 javax.crypto.interfaces.DHPublicKey
未能转换 javax.crypto.KeyAgreement
未能转换 javax.crypto.KeyAgreementSpi
未能转换 javax.crypto.KeyGenerator.generateKey
未能转换 javax.crypto.KeyGenerator.getAlgorithm
未能转换 javax.crypto.KeyGenerator.getProvider
未能转换 javax.crypto.KeyGenerator.init
未能转换 javax.crypto.KeyGeneratorSpi.engineGenerateKey
未能转换 javax.crypto.KeyGeneratorSpi.engineInit
未能转换 javax.crypto.Mac.clone
未能转换 javax.crypto.Mac.doFinal
未能转换 javax.crypto.Mac.getInstance
未能转换 javax.crypto.Mac.getProvider

未能转换 javax.crypto.Mac.Mac
未能转换 javax.crypto.MacSpi.clone
未能转换 javax.crypto.MacSpi.engineDoFinal
未能转换 javax.crypto.MacSpi.MacSpi
未能转换 javax.crypto.NullCipher.NullCipher
未能转换 javax.crypto.SealedObject
未能转换 javax.crypto.SealedObject.encodedParams
未能转换 javax.crypto.SealedObject.getAlgorithm
未能转换 javax.crypto.SealedObject.getObject
未能转换 javax.crypto.SealedObject.SealedObject
未能转换 javax.crypto.SecretKey
未能转换 javax.crypto.SecretKeyFactory.generateSecret
未能转换 javax.crypto.SecretKeyFactory.getAlgorithm
未能转换 javax.crypto.SecretKeyFactory.getKeySpec
未能转换 javax.crypto.SecretKeyFactory.getProvider
未能转换 javax.crypto.SecretKeyFactory.translateKey
未能转换 javax.crypto.SecretKeyFactorySpi.engineGenerateSecret
未能转换 javax.crypto.SecretKeyFactorySpi.engineGetKeySpec
未能转换 javax.crypto.SecretKeyFactorySpi.engineTranslateKey
未能转换 javax.crypto.ShortBufferException.a
未能转换 javax.crypto.spec.DESedeKeySpec.isParityAdjusted
未能转换 javax.crypto.spec.DESKeySpec.isParityAdjusted
未能转换 javax.crypto.spec.DESKeySpec.isWeak
未能转换 javax.crypto.spec.DHGenParameterSpec
未能转换 javax.crypto.spec.DHParameterSpec
未能转换 javax.crypto.spec.DHPrivateKeySpec
未能转换 javax.crypto.spec.DHPublicKeySpec
未能转换 javax.crypto.spec.IvParameterSpec.IvParameterSpec
未能转换 javax.crypto.spec.PBEKeySpec.getPassword
未能转换 javax.crypto.spec.PBEParameterSpec
未能转换 javax.crypto.spec.RC5ParameterSpec
未能转换 javax.crypto.spec.SecretKeySpec.getAlgorithm

Java ejb 错误信息

未能转换 javax.ejb.deployment.AccessControlEntry
未能转换 javax.ejb.deployment.ControlDescriptor
未能转换 javax.ejb.deployment.DeploymentDescriptor
未能转换 javax.ejb.deployment.EntityDescriptor
未能转换 javax.ejb.deployment.SessionDescriptor
未能转换 javax.ejb.EJBContext
未能转换 javax.ejb.EJBContext.getCallerIdentity
未能转换 javax.ejb.EJBContext.getCallerPrincipal
未能转换 javax.ejb.EJBContext.getEJBHome
未能转换 javax.ejb.EJBContext.getEJBLocalHome
未能转换 javax.ejb.EJBContext.getEnvironment
未能转换 javax.ejb.EJBContext.getRollbackOnly
未能转换 javax.ejb.EJBContext.getUserTransaction
未能转换 javax.ejb.EJBContext.isCallerInRole
未能转换 javax.ejb.EJBContext.setRollbackOnly
未能转换 javax.ejb.EJBHome
未能转换 javax.ejb.EJBHome.getEJBMetaData
未能转换 javax.ejb.EJBHome.getHomeHandle
未能转换 javax.ejb.EJBLocalHome
未能转换 javax.ejb.EJBLocalObject
未能转换 javax.ejb.EJBLocalObject.getEJBLocalHome
未能转换 javax.ejb.EJBMetaData
未能转换 javax.ejb.EJBObject
未能转换 javax.ejb.EJBObject.getEJBHome
未能转换 javax.ejb.EJBObject.getHandle
未能转换 javax.ejb.EntityBean.setEntityContext
未能转换 javax.ejb.EntityBean.unsetEntityContext
未能转换 javax.ejb.EntityContext
未能转换 javax.ejb.EntityContext.getEJBLocalObject
未能转换 javax.ejb.EntityContext.getEJBObject
未能转换 javax.ejb.Handle
未能转换 javax.ejb.HomeHandle
未能转换 javax.ejb.MessageDrivenBean.setMessageDrivenContext
未能转换 javax.ejb.MessageDrivenContext
未能转换 javax.ejb.SessionBean.setSessionContext
未能转换 javax.ejb.SessionContext
未能转换 javax.ejb.SessionContext.getEJBLocalObject

未能转换 javax.ejb.SessionContext.getEJBObject

未能转换 javax.ejb.SessionSynchronization

未能转换 javax.ejb.spi.HandleDelegate

Javax.jms 错误信息

未能转换 javax.jms.BytesMessage.readBytes

未能转换 javax.jms.BytesMessage.readUTF

未能转换 javax.jms.BytesMessage.reset

未能转换 javax.jms.BytesMessage.writeBytes

未能转换 javax.jms.BytesMessage.writeObject

未能转换 javax.jms.BytesMessage.writeUTF

未能转换 javax.jms.Connection

未能转换 javax.jms.Connection.close

未能转换 javax.jms.Connection.getClientID

未能转换 javax.jms.Connection.getExceptionListener

未能转换 javax.jms.Connection.getMetaData

未能转换 javax.jms.Connection.setClientID

未能转换 javax.jms.Connection.setExceptionListener

未能转换 javax.jms.Connection.start

未能转换 javax.jms.Connection.stop

未能转换 javax.jms.ConnectionConsumer

未能转换 javax.jms.ConnectionFactory

未能转换 javax.jms.ConnectionMetaData

未能转换 javax.jms.DeliveryMode

未能转换 javax.jms.ExceptionListener

未能转换 javax.jms.IllegalStateException.IllegalStateException

未能转换 javax.jms.InvalidClientIDException

未能转换 javax.jms.InvalidDestinationException.InvalidDestinationException(String)

未能转换 javax.jms.InvalidDestinationException.InvalidDestinationException(String, String)

未能转换 javax.jms.InvalidSelectorException

未能转换 javax.jms.JMSEException.getLinkedException

未能转换 javax.jms.JMSEException.JMSEException(String)

未能转换 javax.jms.JMSEException.JMSEException(String, String)

未能转换 javax.jms.JMSEException.setLinkedException

未能转换 javax.jms.JMSecurityException.JMSecurityException(String)

未能转换 javax.jms.JMSecurityException.JMSecurityException(String, String)

未能转换 javax.jms.MapMessage.getFloat

未能转换 javax.jms.MapMessage.getBytes

未能转换 javax.jms.Message.acknowledge

未能转换 javax.jms.Message.clearProperties

未能转换 javax.jms.Message.getBooleanProperty

未能转换 javax.jms.Message.getByteProperty

未能转换 javax.jms.Message.getDoubleProperty
未能转换 javax.jms.Message.getFloatProperty
未能转换 javax.jms.Message.getIntProperty
未能转换 javax.jms.Message.getJMSCorrelationIDAsBytes
未能转换 javax.jms.Message.getJMSDestination
未能转换 javax.jms.Message.getJMSPriority
未能转换 javax.jms.Message.getJMSRedelivered
未能转换 javax.jms.Message.getJMSReplyTo
未能转换 javax.jms.Message.getJMSTimestamp
未能转换 javax.jms.Message.getJMSType
未能转换 javax.jms.Message.getLongProperty
未能转换 javax.jms.Message.getObjectProperty
未能转换 javax.jms.Message.getPropertyNames
未能转换 javax.jms.Message.getShortProperty
未能转换 javax.jms.Message.getStringProperty
未能转换 javax.jms.Message.propertyExists
未能转换 javax.jms.Message.setBooleanProperty
未能转换 javax.jms.Message.setByteProperty
未能转换 javax.jms.Message.setDoubleProperty
未能转换 javax.jms.Message.setFloatProperty
未能转换 javax.jms.Message.setIntProperty
未能转换 javax.jms.Message.setJMSCorrelationID
未能转换 javax.jms.Message.setJMSCorrelationIDAsBytes
未能转换 javax.jms.Message.setJMSDestination
未能转换 javax.jms.Message.setJMSMessageID
未能转换 javax.jms.Message.setJMSPriority
未能转换 javax.jms.Message.setJMSRedelivered
未能转换 javax.jms.Message.setJMSTimestamp
未能转换 javax.jms.Message.setJMSType
未能转换 javax.jms.Message.setLongProperty
未能转换 javax.jms.Message setObjectProperty
未能转换 javax.jms.Message.setShortProperty
未能转换 javax.jms.Message.setStringProperty
未能转换 javax.jms.MessageConsumer.setMessageListener
未能转换 javax.jms.MessageConsumer.getMessageSelector
未能转换 javax.jms.MessageEOFException.MessageEOFException
未能转换 javax.jms.MessageFormatException
未能转换 javax.jms.MessageListener
未能转换 javax.jms.MessageNotReadableException
未能转换 javax.jms.MessageNotWriteableException

未能转换 javax.jms.MessageProducer.getDisableMessageID
未能转换 javax.jms.MessageProducer.getDisableMessageTimestamp
未能转换 javax.jms.MessageProducer.setDeliveryMode
未能转换 javax.jms.MessageProducer.setDisableMessageID
未能转换 javax.jms.MessageProducer.setDisableMessageTimestamp
未能转换 javax.jms.MessageProducer.setPriority
未能转换 javax.jms.MessageProducer.setTimeToLive
未能转换 javax.jms.QueueConnection
未能转换 javax.jms.QueueConnection.createConnectionConsumer
未能转换 javax.jms.QueueConnection.createQueueSession
未能转换 javax.jms.QueueConnectionFactory
未能转换 javax.jms.QueueConnectionFactory.createQueueConnection
未能转换 javax.jms.QueueConnectionFactory.createQueueConnection(String, String)
未能转换 javax.jms.QueueRequestor
未能转换 javax.jms.QueueSession
未能转换 javax.jms.QueueSession.createBrowser
未能转换 javax.jms.QueueSession.createReceiver
未能转换 javax.jms.QueueBrowser.getMessageSelector
未能转换 javax.jms.QueueConnection
未能转换 javax.jms.QueueConnection.createQueueSession
未能转换 javax.jms.QueueConnectionFactory
未能转换 javax.jms.QueueConnectionFactory.createQueueConnection
未能转换 javax.jms.QueueConnectionFactory.createQueueConnection(String, String)
未能转换 javax.jms.QueueSession
未能转换 javax.jms.ResourceAllocationException.ResourceAllocationException
未能转换 javax.jms.ServerSession
未能转换 javax.jms.ServerSessionPool
未能转换 javax.jms.Session
未能转换 javax.jms.Session.AUTO_ACKNOWLEDGE
未能转换 javax.jms.Session.CLIENT_ACKNOWLEDGE
未能转换 javax.jms.Session.close
未能转换 javax.jms.Session.commit
未能转换 javax.jms.Session.DUPS_OK_ACKNOWLEDGE
未能转换 javax.jms.Session.getMessageListener
未能转换 javax.jms.Session.getTransacted
未能转换 javax.jms.Session.recover
未能转换 javax.jms.Session.rollback
未能转换 javax.jms.Session.run
未能转换 javax.jms.Session.setMessageListener
未能转换 javax.jms.StreamMessage

未能转换 javax.jms.StreamMessage.readBytes
未能转换 javax.jms.StreamMessage.readObject
未能转换 javax.jms.StreamMessage.reset
未能转换 javax.jms.StreamMessage.writeBytes
未能转换 javax.jms.StreamMessage.writeObject
未能转换 javax.jms.TemporaryQueue
未能转换 javax.jms.TemporaryTopic
未能转换 javax.jms.TopicConnection
未能转换 javax.jms.TopicConnection.createConnectionConsumer
未能转换 javax.jms.TopicConnection.createDurableConnectionConsumer
未能转换 javax.jms.TopicConnection.createTopicSession
未能转换 javax.jms.TopicConnectionFactory
未能转换 javax.jms.TopicConnectionFactory.createTopicConnection
未能转换 javax.jms.TopicRequestor
未能转换 javax.jms.TopicSession
未能转换 javax.jms.TopicSession.createDurableSubscriber
未能转换 javax.jms.TopicSession.createSubscriber
未能转换 javax.jms.TopicSession.createTemporaryTopic
未能转换 javax.jms.TopicSession.unsubscribe
未能转换 javax.jms.TopicSubscriber.getNoLocal
未能转换 javax.jms.TransactionInProgressException.TransactionInProgressException
未能转换 javax.jms.TransactionRolledBackException.TransactionRolledBackException
未能转换 javax.jms.XAConnection
未能转换 javax.jms.XAConnectionFactory
未能转换 javax.jms.XAQueueConnection
未能转换 javax.jms.XAQueueConnectionFactory
未能转换 javax.jms.XAQueueSession
未能转换 javax.jms.XASession
未能转换 javax.jms.XATopicConnection
未能转换 javax.jms.XATopicConnectionFactory
未能转换 javax.jms.XATopicSession

Javax.mail 错误信息

未能转换 javax.mail.Address.getType
未能转换 javax.mail.AuthenticationFailedException
未能转换 javax.mail.Authenticator
未能转换 javax.mail.BodyPart.addHeader
未能转换 javax.mail.BodyPart.BodyPart
未能转换 javax.mail.BodyPart.getAllHeaders
未能转换 javax.mail.BodyPart.getContent
未能转换 javax.mail.BodyPart.getContentType
未能转换 javax.mail.BodyPart.getDataHandler
未能转换 javax.mail.BodyPart.getDescription
未能转换 javax.mail.BodyPart.getDisposition
未能转换 javax.mail.BodyPart.getHeader
未能转换 javax.mail.BodyPart.getLineCount
未能转换 javax.mail.BodyPart.getMatchingHeaders
未能转换 javax.mail.BodyPart.getParent
未能转换 javax.mail.BodyPart.getSize
未能转换 javax.mail.BodyPart.isMimeType
未能转换 javax.mail.BodyPart.parent
未能转换 javax.mail.BodyPart.removeHeader
未能转换 javax.mail.BodyPart.setContent(Multipart)
未能转换 javax.mail.BodyPart.setContent(Object, String)
未能转换 javax.mail.BodyPart.setDataHandler
未能转换 javax.mail.BodyPart.setDescription
未能转换 javax.mail.BodyPart.setDisposition
未能转换 javax.mail.BodyPart.setFileName
未能转换 javax.mail.BodyPart.setHeader
未能转换 javax.mail.BodyPart.writeTo
未能转换 javax.mail.event.ConnectionAdapter
未能转换 javax.mail.event.ConnectionEvent
未能转换 javax.mail.event.ConnectionListener
未能转换 javax.mail.event.FolderAdapter
未能转换 javax.mail.event.FolderEvent
未能转换 javax.mail.event.FolderListener
未能转换 javax.mail.event.MailEvent
未能转换 javax.mail.event.MessageChangedEvent
未能转换 javax.mail.event.MessageChangedListener
未能转换 javax.mail.event.MessageCountAdapter

未能转换 javax.mail.event.MessageCountEvent
未能转换 javax.mail.event.MessageCountListener
未能转换 javax.mail.event.StoreEvent
未能转换 javax.mail.event.StoreListener
未能转换 javax.mail.event.TransportAdapter
未能转换 javax.mail.event.TransportEvent
未能转换 javax.mail.event.TransportListener
未能转换 javax.mail.FetchProfile
未能转换 javax.mail.FetchProfile.Item
未能转换 javax.mail.Flags
未能转换 javax.mail.Flags.Flag
未能转换 javax.mail.Folder
未能转换 javax.mail.FolderClosedException
未能转换 javax.mail.FolderNotFoundException
未能转换 javax.mail.Header.Header
未能转换 javax.mail.IllegalWriteException
未能转换 javax.mail.internet.AddressException
未能转换 javax.mail.internet.AddressException.getPos
未能转换 javax.mail.internet.AddressException.getRef
未能转换 javax.mail.internet.AddressException.pos
未能转换 javax.mail.internet.AddressException.ref
未能转换 javax.mail.internet.ContentDisposition
未能转换 javax.mail.internet.ContentType
未能转换 javax.mail.internet.HeaderTokenizer
未能转换 javax.mail.internet.HeaderTokenizer.Token
未能转换 javax.mail.internet.InternetAddress.clone
未能转换 javax.mail.internet.InternetAddress.encodedPersonal
未能转换 javax.mail.internet.InternetHeaders.getHeader
未能转换 javax.mail.internet.InternetAddress.getLocalAddress
未能转换 javax.mail.internet.InternetAddress.getPersonal
未能转换 javax.mail.internet.InternetAddress.InternetAddress
未能转换 javax.mail.internet.InternetAddress.personal
未能转换 javax.mail.internet.InternetAddress.setAddress
未能转换 javax.mail.internet.InternetAddress.setPersonal
未能转换 javax.mail.internet.InternetHeaders.addHeader
未能转换 javax.mail.internet.InternetHeaders.getMatchingHeaderLines
未能转换 javax.mail.internet.InternetHeaders.getMatchingHeaders
未能转换 javax.mail.internet.InternetHeaders.getNonMatchingHeaderLines
未能转换 javax.mail.internet.InternetHeaders.getNonMatchingHeaders
未能转换 javax.mail.internet.InternetHeaders.InternetHeaders

未能转换 javax.mail.internet.InternetHeaders.load
未能转换 javax.mail.internet.MailDateFormat
未能转换 javax.mail.internet.MimeBodyPart.addHeader
未能转换 javax.mail.internet.MimeBodyPart.addHeaderLine
未能转换 javax.mail.internet.MimeBodyPart.content
未能转换 javax.mail.internet.MimeBodyPart.contentStream
未能转换 javax.mail.internet.MimeBodyPart.dh
未能转换 javax.mail.internet.MimeBodyPart.getAllHeaderLines
未能转换 javax.mail.internet.MimeBodyPart.getAllHeaders
未能转换 javax.mail.internet.MimeBodyPart.getContent
未能转换 javax.mail.internet.MimeBodyPart.getContentID
未能转换 javax.mail.internet.MimeBodyPart.getContentLanguage
未能转换 javax.mail.internet.MimeBodyPart.getContentMD5
未能转换 javax.mail.internet.MimeBodyPart.getContentStream
未能转换 javax.mail.internet.MimeBodyPart.getContentType
未能转换 javax.mail.internet.MimeBodyPart.getDataHandler
未能转换 javax.mail.internet.MimeBodyPart.getDescription
未能转换 javax.mail.internet.MimeBodyPart.getDisposition
未能转换 javax.mail.internet.MimeBodyPart.getEncoding
未能转换 javax.mail.internet.MimeBodyPart.getHeader(String)
未能转换 javax.mail.internet.MimeBodyPart.getHeader(String, String)
未能转换 javax.mail.internet.MimeBodyPart.getLineCount
未能转换 javax.mail.internet.MimeBodyPart.getMatchingHeaderLines
未能转换 javax.mail.internet.MimeBodyPart.getMatchingHeaders
未能转换 javax.mail.internet.MimeBodyPart.getNonMatchingHeaderLines
未能转换 javax.mail.internet.MimeBodyPart.getNonMatchingHeaders
未能转换 javax.mail.internet.MimeBodyPart.getRawInputStream
未能转换 javax.mail.internet.MimeBodyPart.getSize
未能转换 javax.mail.internet.MimeBodyPart.headers
未能转换 javax.mail.internet.MimeBodyPart.isMimeType
未能转换 javax.mail.internet.MimeBodyPart.MimeBodyPart
未能转换 javax.mail.internet.MimeBodyPart.MimeBodyPart(InputStream)
未能转换 javax.mail.internet.MimeBodyPart.MimeBodyPart(InternetHeaders, byte[])
未能转换 javax.mail.internet.MimeBodyPart.removeHeader
未能转换 javax.mail.internet.MimeBodyPart.setContent(Multipart)
未能转换 javax.mail.internet.MimeBodyPart.setContent(Object, String)
未能转换 javax.mail.internet.MimeBodyPart.setContentLanguage
未能转换 javax.mail.internet.MimeBodyPart.setContentMD5
未能转换 javax.mail.internet.MimeBodyPart.setDataHandler
未能转换 javax.mail.internet.MimeBodyPart.setDescription

未能转换 javax.mail.internet.MimeBodyPart.setDisposition
未能转换 javax.mail.internet.MimeBodyPart.setHeader
未能转换 javax.mail.internet.MimeBodyPart.setText
未能转换 javax.mail.internet.MimeBodyPart.updateHeaders
未能转换 javax.mail.internet.MimeBodyPart.writeTo
未能转换 javax.mail.internet.MimeMessage.addHeader
未能转换 javax.mail.internet.MimeMessage.addHeaderLine
未能转换 javax.mail.internet.MimeMessage.addRecipients
未能转换 javax.mail.internet.MimeMessage.content
未能转换 javax.mail.internet.MimeMessage.contentStream
未能转换 javax.mail.internet.MimeMessage.createInternetHeaders
未能转换 javax.mail.internet.MimeMessage.dh
未能转换 javax.mail.internet.MimeMessage.flags
未能转换 javax.mail.internet.MimeMessage.getAllHeaderLines
未能转换 javax.mail.internet.MimeMessage.getAllRecipients
未能转换 javax.mail.internet.MimeMessage.getContentID
未能转换 javax.mail.internet.MimeMessage.getContentLanguage
未能转换 javax.mail.internet.MimeMessage.getContentMD5
未能转换 javax.mail.internet.MimeMessage.getContentStream
未能转换 javax.mail.internet.MimeMessage.getContentType
未能转换 javax.mail.internet.MimeMessage.getDataHandler
未能转换 javax.mail.internet.MimeMessage.getDescription
未能转换 javax.mail.internet.MimeMessage.getDisposition
未能转换 javax.mail.internet.MimeMessage.getEncoding
未能转换 javax.mail.internet.MimeMessage.getFileName
未能转换 javax.mail.internet.MimeMessage.getFlags
未能转换 javax.mail.internet.MimeMessage.getFrom
未能转换 javax.mail.internet.MimeMessage.getHeader
未能转换 javax.mail.internet.MimeMessage.getInputStream
未能转换 javax.mail.internet.MimeMessage.getLineCount
未能转换 javax.mail.internet.MimeMessage.getMatchingHeaderLines
未能转换 javax.mail.internet.MimeMessage.getMatchingHeaders
未能转换 javax.mail.internet.MimeMessage.getMessageID
未能转换 javax.mail.internet.MimeMessage.getNonMatchingHeaderLines
未能转换 javax.mail.internet.MimeMessage.getRawInputStream
未能转换 javax.mail.internet.MimeMessage.getReceivedDate
未能转换 javax.mail.internet.MimeMessage.getRecipients
未能转换 javax.mail.internet.MimeMessage.getReplyTo
未能转换 javax.mail.internet.MimeMessage.getSentDate
未能转换 javax.mail.internet.MimeMessage.getSize

未能转换 javax.mail.internet.MimeMessage.headers
未能转换 javax.mail.internet.MimeMessage.isMimeType
未能转换 javax.mail.internet.MimeMessage.isSet
未能转换 javax.mail.internet.MimeMessage.MimeMessage
未能转换 javax.mail.internet.MimeMessage.modified
未能转换 javax.mail.internet.MimeMessage.parse
未能转换 javax.mail.internet.MimeMessage.RecipientType
未能转换 javax.mail.internet.MimeMessage.reply
未能转换 javax.mail.internet.MimeMessage.saveChanges
未能转换 javax.mail.internet.MimeMessage.saved
未能转换 javax.mail.internet.MimeMessage.setContent
未能转换 javax.mail.internet.MimeMessage.setContentID
未能转换 javax.mail.internet.MimeMessage.setContentLanguage
未能转换 javax.mail.internet.MimeMessage.setContentMD5
未能转换 javax.mail.internet.MimeMessage.setDataHandler
未能转换 javax.mail.internet.MimeMessage.setDescription
未能转换 javax.mail.internet.MimeMessage.setDisposition
未能转换 javax.mail.internet.MimeMessage.setFileName
未能转换 javax.mail.internet.MimeMessage.setFlags
未能转换 javax.mail.internet.MimeMessage.setFrom
未能转换 javax.mail.internet.MimeMessage.setRecipients
未能转换 javax.mail.internet.MimeMessage.setSentDate
未能转换 javax.mail.internet.MimeMessage.setSubject
未能转换 javax.mail.internet.MimeMessage.setText
未能转换 javax.mail.internet.MimeMessage.updateHeaders
未能转换 javax.mail.internet.MimeMessage.writeTo
未能转换 javax.mail.internet.MimeMultipart.createInternetHeaders
未能转换 javax.mail.internet.MimeMultipart.createMimeBodyPart
未能转换 javax.mail.internet.MimeMultipart.ds
未能转换 javax.mail.internet.MimeMultipart.getBodyPart
未能转换 javax.mail.internet.MimeMultipart.MimeMultipart(DataSource)
未能转换 javax.mail.internet.MimeMultipart.MimeMultipart(String)
未能转换 javax.mail.internet.MimeMultipart.parse
未能转换 javax.mail.internet.MimeMultipart.parsed
未能转换 javax.mail.internet.MimeMultipart.setSubType
未能转换 javax.mail.internet.MimeMultipart.updateHeaders
未能转换 javax.mail.internet.MimeMultipart.writeTo
未能转换 javax.mail.internet.MimePart.getContentID
未能转换 javax.mail.internet.MimePart.getContentLanguage
未能转换 javax.mail.internet.MimePart.getContentMD5

未能转换 javax.mail.internet.MimePart.getMatchingHeaderLines
未能转换 javax.mail.internet.MimePart.getNonMatchingHeaderLines
未能转换 javax.mail.internet.MimePart.setContentLanguage
未能转换 javax.mail.internet.MimePart.setContentMD5
未能转换 javax.mail.internet.MimePart.setText
未能转换 javax.mail.internet.MimePartDataSource
未能转换 javax.mail.internet.MimeUtility
未能转换 javax.mail.internet.NewsAddress
未能转换 javax.mail.internet.ParameterList
未能转换 javax.mail.internet.ParseException
未能转换 javax.mail.internet.SharedInputStream
未能转换 javax.mail.Message.addHeader
未能转换 javax.mail.Message.expunged
未能转换 javax.mail.Message.folder
未能转换 javax.mail.Message.getAllRecipients
未能转换 javax.mail.Message.getContentType
未能转换 javax.mail.Message.getDataHandler
未能转换 javax.mail.Message.getDescription
未能转换 javax.mail.Message.getDisposition
未能转换 javax.mail.Message.getFileName
未能转换 javax.mail.Message.getFlags
未能转换 javax.mail.Message.getFolder
未能转换 javax.mail.Message.getFrom
未能转换 javax.mail.Message.getInputStream
未能转换 javax.mail.Message.getLineCount
未能转换 javax.mail.Message.getMatchingHeaders
未能转换 javax.mail.Message.getMessageNumber
未能转换 javax.mail.Message.getNonMatchingHeaders
未能转换 javax.mail.Message.getReceivedDate
未能转换 javax.mail.Message.getRecipients
未能转换 javax.mail.Message.getReplyTo
未能转换 javax.mail.Message.getSentDate
未能转换 javax.mail.Message.getSize
未能转换 javax.mail.Message.isExpunged
未能转换 javax.mail.Message.isMimeType
未能转换 javax.mail.Message.isSet
未能转换 javax.mail.Message.match
未能转换 javax.mail.Message.Message
未能转换 javax.mail.Message.msgnum
未能转换 javax.mail.Message.RecipientType.readResolve

未能转换 javax.mail.Message.RecipientType.type
未能转换 javax.mail.Message.reply
未能转换 javax.mail.Message.saveChanges
未能转换 javax.mail.Message.session
未能转换 javax.mail.Message.setContent
未能转换 javax.mail.Message.setDataHandler
未能转换 javax.mail.Message.setDescription
未能转换 javax.mail.Message.setDisposition
未能转换 javax.mail.Message.setExpunged
未能转换 javax.mail.Message.setFileName
未能转换 javax.mail.Message.setFlag
未能转换 javax.mail.Message.setFlags
未能转换 javax.mail.Message.setFrom
未能转换 javax.mail.Message.setMessageNumber
未能转换 javax.mail.Message.setSentDate
未能转换 javax.mail.Message.writeTo
未能转换 javax.mail.MessageAware
未能转换 javax.mail.MessageContext
未能转换 javax.mail.MessageRemovedException
未能转换 javax.mail.MessagingException.setNextException
未能转换 javax.mail.MethodNotSupportedException
未能转换 javax.mail.Multipart.contentType
未能转换 javax.mail.Multipart.getContentType
未能转换 javax.mail.Multipart.getParent
未能转换 javax.mail.Multipart.parent
未能转换 javax.mail.Multipart.setMultipartDataSource
未能转换 javax.mail.Multipart.setParent
未能转换 javax.mail.Multipart.writeTo
未能转换 javax.mail.MultipartDataSource
未能转换 javax.mail.NoSuchProviderException
未能转换 javax.mail.Part.addHeader
未能转换 javax.mail.Part.ATTACHMENT
未能转换 javax.mail.Part.getContentType
未能转换 javax.mail.Part.getDataHandler
未能转换 javax.mail.Part.getDescription
未能转换 javax.mail.Part.getDisposition
未能转换 javax.mail.Part.getFileName
未能转换 javax.mail.Part.getInputStream
未能转换 javax.mail.Part.getLineCount
未能转换 javax.mail.Part.getMatchingHeaders

未能转换 javax.mail.Part.getNonMatchingHeaders

未能转换 javax.mail.Part.getSize

未能转换 javax.mail.Part.INLINE

未能转换 javax.mail.Part.isMimeType

未能转换 javax.mail.Part.setContent

未能转换 javax.mail.Part.setDataHandler

未能转换 javax.mail.Part.setDescription

未能转换 javax.mail.Part.setDisposition

未能转换 javax.mail.Part.setFileName

未能转换 javax.mail.Part.writeTo

未能转换 javax.mail.PasswordAuthentication

未能转换 javax.mail.Provider

未能转换 javax.mail.Provider.Type

未能转换 javax.mail.ReadOnlyFolderException

未能转换 javax.mail.search.AddressStringTerm

未能转换 javax.mail.search.AddressTerm

未能转换 javax.mail.search.AndTerm

未能转换 javax.mail.search.BodyTerm

未能转换 javax.mail.search.ComparisonTerm

未能转换 javax.mail.search.DateTerm

未能转换 javax.mail.search.FlagTerm

未能转换 javax.mail.search.FromStringTerm

未能转换 javax.mail.search.FromTerm

未能转换 javax.mail.search.HeaderTerm

未能转换 javax.mail.search.IntegerComparisonTerm

未能转换 javax.mail.search.MessageIDTerm

未能转换 javax.mail.search.MessageNumberTerm

未能转换 javax.mail.search.NotTerm

未能转换 javax.mail.search.OrTerm

未能转换 javax.mail.search.ReceivedDateTerm

未能转换 javax.mail.search.RecipientStringTerm

未能转换 javax.mail.search.RecipientTerm

未能转换 javax.mail.search.SearchException

未能转换 javax.mail.search.SearchTerm

未能转换 javax.mail.search.SentDateTerm

未能转换 javax.mail.search.SizeTypeTerm

未能转换 javax.mail.search.StringTerm

未能转换 javax.mail.search.SubjectTerm

未能转换 javax.mail.SendFailedException.getInvalidAddresses

未能转换 javax.mail.SendFailedException.getValidSentAddresses

未能转换 javax.mail.SendFailedException.getValidUnsentAddresses
未能转换 javax.mail.SendFailedException.invalid
未能转换 javax.mail.SendFailedException.validSent
未能转换 javax.mail.SendFailedException.validUnsent
未能转换 javax.mail.Service
未能转换 javax.mail.Session
未能转换 javax.mail.Store
未能转换 javax.mail.StoreClosedException
未能转换 javax.mail.Transport.addTransportListener
未能转换 javax.mail.Transport.notifyTransportListeners
未能转换 javax.mail.Transport.removeTransportListener
未能转换 javax.mail.Transport.send
未能转换 javax.mail.Transport.Transport
未能转换 javax.mail.UIDFolder
未能转换 javax.mail.UIDFolder.FetchProfileItem
未能转换 javax.mail.URLName.fullURL
未能转换 javax.mail.URLName.getFile
未能转换 javax.mail.URLName.getRef
未能转换 javax.mail.URLName.parseString
未能转换 javax.mail.URLName.URLName

Java.naming 错误信息

未能转换 javax.naming.BinaryRefAddr
未能转换 javax.naming.BinaryRefAddr.BinaryRefAddr(String, byte[])
未能转换 javax.naming.BinaryRefAddr.BinaryRefAddr(String, byte[], int, int)
未能转换 javax.naming.Binding
未能转换 javax.naming.Binding.Binding(String, Object)
未能转换 javax.naming.Binding.Binding(String, Object, boolean)
未能转换 javax.naming.Binding.Binding(String, String, Object)
未能转换 javax.naming.Binding.Binding(String, String, Object, boolean)
未能转换 javax.naming.Binding.setObject
未能转换 javax.naming.CannotProceedException.altName
未能转换 javax.naming.CannotProceedException.altNameCtx
未能转换 javax.naming.CannotProceedException.environment
未能转换 javax.naming.CannotProceedException.getAltName
未能转换 javax.naming.CannotProceedException.getAltNameCtx
未能转换 javax.naming.CannotProceedException.getEnvironment
未能转换 javax.naming.CannotProceedException.getRemainingNewName
未能转换 javax.naming.CannotProceedException.remainingNewName
未能转换 javax.naming.CannotProceedException.setAltName
未能转换 javax.naming.CannotProceedException.setAltNameCtx
未能转换 javax.naming.CannotProceedException.setEnvironment
未能转换 javax.naming.CannotProceedException.setRemainingNewName
未能转换 javax.naming.CompositeName
未能转换 javax.naming.CompositeName.CompositeName
未能转换 javax.naming.CompoundName
未能转换 javax.naming.CompoundName.CompoundName(Enumeration, Properties)
未能转换 javax.naming.CompoundName.CompoundName(String, Properties)
未能转换 javax.naming.CompoundName.impl
未能转换 javax.naming.CompoundName.mySyntax
未能转换 javax.naming.Context.<EnvironmentProperty>
未能转换 javax.naming.Context.addToEnvironment
未能转换 javax.naming.Context.bind(Name, Object)
未能转换 javax.naming.Context.bind(String, Object)
未能转换 javax.naming.Context.composeName
未能转换 javax.naming.Context.createSubcontext
未能转换 javax.naming.Context.destroySubcontext
未能转换 javax.naming.Context.getEnvironment
未能转换 javax.naming.Context.getNameInNamespace

未能转换 javax.naming.Context.getNameParser
未能转换 javax.naming.Context.list
未能转换 javax.naming.Context.listBindings
未能转换 javax.naming.Context.lookup
未能转换 javax.naming.Context.lookupLink
未能转换 javax.naming.Context.rebind
未能转换 javax.naming.Context.removeFromEnvironment
未能转换 javax.naming.Context.rename
未能转换 javax.naming.Context.unbind
未能转换 javax.naming.directory.Attribute
未能转换 javax.naming.directory.Attribute.clone
未能转换 javax.naming.directory.Attribute.get
未能转换 javax.naming.directory.Attribute.getAttributeDefinition
未能转换 javax.naming.directory.Attribute.getAttributeSyntaxDefinition
未能转换 javax.naming.directory.Attribute.getId
未能转换 javax.naming.directory.Attribute.isOrdered
未能转换 javax.naming.directory.Attribute.serialVersionUID
未能转换 javax.naming.directory.AttributeModificationException.getUnexecutedModifications
未能转换 javax.naming.directory.AttributeModificationException.setUnexecutedModifications
未能转换 javax.naming.directory.Attributes
未能转换 javax.naming.directory.Attributes.clone
未能转换 javax.naming.directory.Attributes.isCaseIgnored
未能转换 javax.naming.directory.Attributes.put
未能转换 javax.naming.directory.Attributes.remove
未能转换 javax.naming.directory.BasicAttribute
未能转换 javax.naming.directory.BasicAttribute.attrID
未能转换 javax.naming.directory.BasicAttribute.BasicAttribute(String)
未能转换 javax.naming.directory.BasicAttribute.BasicAttribute(String, boolean)
未能转换 javax.naming.directory.BasicAttribute.BasicAttribute(String, Object)
未能转换 javax.naming.directory.BasicAttribute.BasicAttribute(String, Object, boolean)
未能转换 javax.naming.directory.BasicAttribute.clone
未能转换 javax.naming.directory.BasicAttribute.get
未能转换 javax.naming.directory.BasicAttribute.getAttributeDefinition
未能转换 javax.naming.directory.BasicAttribute.getAttributeSyntaxDefinition
未能转换 javax.naming.directory.BasicAttribute.getId
未能转换 javax.naming.directory.BasicAttribute.isOrdered
未能转换 javax.naming.directory.BasicAttribute.ordered
未能转换 javax.naming.directory.BasicAttribute.values
未能转换 javax.naming.directory.BasicAttributes
未能转换 javax.naming.directory.BasicAttributes.BasicAttributes

未能转换 javax.naming.directory.BasicAttributes.clone
未能转换 javax.naming.directory.BasicAttributes.isCaseIgnored
未能转换 javax.naming.directory.BasicAttributes.put
未能转换 javax.naming.directory.BasicAttributes.remove
未能转换 javax.naming.directory.DirContext
未能转换 javax.naming.directory.DirContext.<OperationType>
未能转换 javax.naming.directory.DirContext.bind(Name, Object, Attributes)
未能转换 javax.naming.directory.DirContext.bind(String, Object, Attributes)
未能转换 javax.naming.directory.DirContext.createSubcontext
未能转换 javax.naming.directory.DirContext.getAttributes
未能转换 javax.naming.directory.DirContext.getSchema
未能转换 javax.naming.directory.DirContext.getSchemaClassDefinition
未能转换 javax.naming.directory.DirContext.modifyAttributes
未能转换 javax.naming.directory.DirContext.rebind
未能转换 javax.naming.directory.DirContext.search
未能转换 javax.naming.directory.InitialDirContext.bind
未能转换 javax.naming.directory.InitialDirContext.createSubcontext
未能转换 javax.naming.directory.InitialDirContext.getAttributes
未能转换 javax.naming.directory.InitialDirContext.getSchema
未能转换 javax.naming.directory.InitialDirContext.getSchemaClassDefinition
未能转换 javax.naming.directory.InitialDirContext.InitialDirContext(boolean)
未能转换 javax.naming.directory.InitialDirContext.InitialDirContext(Hashtable)
未能转换 javax.naming.directory.InitialDirContext.modifyAttributes
未能转换 javax.naming.directory.InitialDirContext.rebind
未能转换 javax.naming.directory.InitialDirContext.search
未能转换 javax.naming.directory.SearchControls.getDerefLinkFlag
未能转换 javax.naming.directory.SearchControls.getReturningObjFlag
未能转换 javax.naming.directory.SearchControls.SearchControls
未能转换 javax.naming.directory.SearchControls.setDerefLinkFlag
未能转换 javax.naming.directory.SearchControls.setReturningObjFlag
未能转换 javax.naming.directory.SearchResult.getAttributes
未能转换 javax.naming.directory.SearchResult.SearchResult(String, Object, Attributes)
未能转换 javax.naming.directory.SearchResult.SearchResult(String, Object, Attributes, boolean)
未能转换 javax.naming.directory.SearchResult.SearchResult(String, String, Object, Attributes)
未能转换 javax.naming.directory.SearchResult.SearchResult(String, String, Object, Attributes, boolean)
未能转换 javax.naming.directory.SearchResult.setAttributes
未能转换 javax.naming.event.EventContext
未能转换 javax.naming.event.EventDirContext
未能转换 javax.naming.event.NamespaceChangeListener
未能转换 javax.naming.event.NamingEvent

未能转换 javax.naming.event.NamingExceptionEvent
未能转换 javax.naming.event.NamingListener
未能转换 javax.naming.event.ObjectChangeListener
未能转换 javax.naming.InitialContext.addToEnvironment
未能转换 javax.naming.InitialContext.bind
未能转换 javax.naming.InitialContext.composeName
未能转换 javax.naming.InitialContext.createSubcontext
未能转换 javax.naming.InitialContext.destroySubcontext
未能转换 javax.naming.InitialContext.getDefaultInitCtx
未能转换 javax.naming.InitialContext.getEnvironment
未能转换 javax.naming.InitialContext.getNameInNamespace
未能转换 javax.naming.InitialContext.getNameParser
未能转换 javax.naming.InitialContext.getURLOrDefaultInitCtx
未能转换 javax.naming.InitialContext.getDefDefault
未能转换 javax.naming.InitialContext.init
未能转换 javax.naming.InitialContext.InitialContext
未能转换 javax.naming.InitialContext.InitialContext(boolean)
未能转换 javax.naming.InitialContext.InitialContext(Hashtable)
未能转换 javax.naming.InitialContext.list
未能转换 javax.naming.InitialContext.listBindings
未能转换 javax.naming.InitialContext.lookup
未能转换 javax.naming.InitialContext.lookupLink
未能转换 javax.naming.InitialContext.myProps
未能转换 javax.naming.InitialContext.rebind
未能转换 javax.naming.InitialContext.removeFromEnvironment
未能转换 javax.naming.InitialContext.rename
未能转换 javax.naming.InitialContext.unbind
未能转换 javax.naming.ldap.Control
未能转换 javax.naming.ldap.ControlFactory
未能转换 javax.naming.ldap.ExtendedRequest
未能转换 javax.naming.ldap.ExtendedResponse
未能转换 javax.naming.ldap.HasControls
未能转换 javax.naming.ldap.InitialLdapContext
未能转换 javax.naming.ldap.InitialLdapContext.extendedOperation
未能转换 javax.naming.ldap.InitialLdapContext.getConnectControls
未能转换 javax.naming.ldap.InitialLdapContext.getRequestControls
未能转换 javax.naming.ldap.InitialLdapContext.getResponseControls
未能转换 javax.naming.ldap.InitialLdapContext.InitialLdapContext
未能转换 javax.naming.ldap.InitialLdapContext.newInstance
未能转换 javax.naming.ldap.InitialLdapContext.reconnect

未能转换 javax.naming.ldap.InitialLdapContext.setRequestControls
未能转换 javax.naming.ldap.LdapContext
未能转换 javax.naming.ldap.LdapContext.CONTROL_FACTORIES
未能转换 javax.naming.ldap.LdapContext.extendedOperation
未能转换 javax.naming.ldap.LdapContext.getConnectControls
未能转换 javax.naming.ldap.LdapContext.getRequestControls
未能转换 javax.naming.ldap.LdapContext.getResponseControls
未能转换 javax.naming.ldap.LdapContext.newInstance
未能转换 javax.naming.ldap.LdapContext.reconnect
未能转换 javax.naming.ldap.LdapContext.setRequestControls
未能转换 javax.naming.ldap.LdapReferralException
未能转换 javax.naming.ldap.UnsolicitedNotification
未能转换 javax.naming.ldap.UnsolicitedNotificationEvent
未能转换 javax.naming.ldap.UnsolicitedNotificationListener
未能转换 javax.naming.LinkException.getLinkRemainingName
未能转换 javax.naming.LinkException.getLinkResolvedName
未能转换 javax.naming.LinkException.getLinkResolvedObj
未能转换 javax.naming.LinkException.linkRemainingName
未能转换 javax.naming.LinkException.linkResolvedName
未能转换 javax.naming.LinkException.linkResolvedObj
未能转换 javax.naming.LinkException.setLinkRemainingName
未能转换 javax.naming.LinkException.setLinkResolvedName
未能转换 javax.naming.LinkException.setLinkResolvedObj
未能转换 javax.naming.LinkException.toString
未能转换 javax.naming.LinkRef
未能转换 javax.naming.Name
未能转换 javax.naming.NameClassPair.isRelative
未能转换 javax.naming.NameClassPair.NameClassPair(String, String)
未能转换 javax.naming.NameClassPair.NameClassPair(String, String, boolean)
未能转换 javax.naming.NameClassPair.setClassName
未能转换 javax.naming.NameClassPair.setRelative
未能转换 javax.naming.NameParser
未能转换 javax.naming.NamingEnumeration.close
未能转换 javax.naming.NamingEnumeration.hasMore
未能转换 javax.naming.NamingEnumeration.next
未能转换 javax.naming.NamingException.appendRemainingComponent
未能转换 javax.naming.NamingException.appendRemainingName
未能转换 javax.naming.NamingException.getRemainingName
未能转换 javax.naming.NamingException.getResolvedName
未能转换 javax.naming.NamingException.getResolvedObj

未能转换 javax.naming.NamingException.printStackTrace
未能转换 javax.naming.NamingException.remainingName
未能转换 javax.naming.NamingException.resolvedName
未能转换 javax.naming.NamingException.resolvedObj
未能转换 javax.naming.NamingException.rootException
未能转换 javax.naming.NamingException.setRemainingName
未能转换 javax.naming.NamingException.setResolvedName
未能转换 javax.naming.NamingException.setResolvedObj
未能转换 javax.naming.NamingException.setRootCause
未能转换 javax.naming.NamingException.toString
未能转换 javax.naming.RefAddr
未能转换 javax.naming.RefAddr.addrType
未能转换 javax.naming.RefAddr.RefAddr
未能转换 javax.naming.Reference.addrs
未能转换 javax.naming.Reference.classFactory
未能转换 javax.naming.Reference.classFactoryLocation
未能转换 javax.naming.Reference.className
未能转换 javax.naming.Reference.get
未能转换 javax.naming.Reference.getClassName
未能转换 javax.naming.Reference.getFactoryClassLocation
未能转换 javax.naming.Reference.getFactoryClassName
未能转换 javax.naming.Reference.Reference
未能转换 javax.naming.Referenceable
未能转换 javax.naming.ReferralException.getReferralContext
未能转换 javax.naming.ReferralException.getReferralInfo
未能转换 javax.naming.ReferralException.retryReferral
未能转换 javax.naming.ReferralException.skipReferral
未能转换 javax.naming.spi.DirectoryManager
未能转换 javax.naming.spi.DirObjectFactory
未能转换 javax.naming.spi.DirStateFactory
未能转换 javax.naming.spi.DirStateFactory.Result
未能转换 javax.naming.spi.InitialContextFactory
未能转换 javax.naming.spi.InitialContextBuilderFactory
未能转换 javax.naming.spi.NamingManager
未能转换 javax.naming.spi.ObjectFactory
未能转换 javax.naming.spi.ObjectFactoryBuilder
未能转换 javax.naming.spi.Resolver
未能转换 javax.naming.spi.ResolveResult
未能转换 javax.naming.spi.StateFactory
未能转换 javax.naming.StringRefAddr

未能转换 javax.naming.StringRefAddr.StringRefAddr

Javax.rmi 错误信息

未能转换 javax.rmi.CORBA.ClassDesc

未能转换 javax.rmi.CORBA.PortableRemoteObjectDelegate

未能转换 javax.rmi.CORBA.Stub

未能转换 javax.rmi.CORBA.StubDelegate

未能转换 javax.rmi.CORBA.Tie

未能转换 javax.rmi.CORBA.Util

未能转换 javax.rmi.CORBA.UtilDelegate

未能转换 javax.rmi.CORBA.ValueHandler

未能转换 javax.rmi.PortableRemoteObject

未能转换 javax.rmi.PortableRemoteObject.narrow

Java.security 错误信息

未能转换 javax.security.auth.AuthPermission
未能转换 javax.security.auth.callback.ChoiceCallback.ChoiceCallback
未能转换 javax.security.auth.callback.ChoiceCallback.getPrompt
未能转换 javax.security.auth.callback.LanguageCallback.LanguageCallback
未能转换 javax.security.auth.callback.NameCallback.getDefaultName
未能转换 javax.security.auth.callback.NameCallback.getPrompt
未能转换 javax.security.auth.callback.NameCallback.NameCallback(String)
未能转换 javax.security.auth.callback.NameCallback.NameCallback(String, String)
未能转换 javax.security.auth.callback.PasswordCallback.getPrompt
未能转换 javax.security.auth.callback.PasswordCallback.isEchoOn
未能转换 javax.security.auth.callback.PasswordCallback.PasswordCallback
未能转换 javax.security.auth.callback.UnsupportedCallbackException
未能转换 javax.security.auth.callback.UnsupportedCallbackException.getCallback
未能转换 javax.security.auth.callback.UnsupportedCallbackException.UnsupportedCallbackException
未能转换 javax.security.auth.Destroyable.isDestroyed
未能转换 javax.security.auth.DestroyFailedException
未能转换 javax.security.auth.DestroyFailedException.DestroyFailedException
未能转换 javax.security.auth.login.AccountExpiredException
未能转换 javax.security.auth.login.AccountExpiredException.AccountExpiredException
未能转换 javax.security.auth.login.AppConfigurationEntry
未能转换 javax.security.auth.login.AppConfigurationEntry.AppConfigurationEntry
未能转换 javax.security.auth.login.AppConfigurationEntry.getControlFlag
未能转换 javax.security.auth.login.AppConfigurationEntry.getLoginModuleName
未能转换 javax.security.auth.login.Configuration
未能转换 javax.security.auth.login.Configuration.Configuration
未能转换 javax.security.auth.login.Configuration.refresh
未能转换 javax.security.auth.login.Configuration.setConfiguration
未能转换 javax.security.auth.login.CredentialExpiredException
未能转换 javax.security.auth.login.CredentialExpiredException.CredentialExpiredException
未能转换 javax.security.auth.login.FailedLoginException
未能转换 javax.security.auth.login.FailedLoginException.FailedLoginException
未能转换 javax.security.auth.login.LoginContext.getSubject
未能转换 javax.security.auth.login.LoginContext.login
未能转换 javax.security.auth.login.LoginContext.LoginContext
未能转换 javax.security.auth.login.LoginContext.logout
未能转换 javax.security.auth.login.LoginException
未能转换 javax.security.auth.login.LoginException.LoginException

未能转换 javax.security.auth.Policy
未能转换 javax.security.auth.Policy.getPolicy
未能转换 javax.security.auth.Policy.Policy
未能转换 javax.security.auth.Policy.refresh
未能转换 javax.security.auth.Policy.setPolicy
未能转换 javax.security.auth.PrivateCredentialPermission
未能转换 javax.security.auth.PrivateCredentialPermission.getActions
未能转换 javax.security.auth.PrivateCredentialPermission.getCredentialClass
未能转换 javax.security.auth.PrivateCredentialPermission.getPrincipals
未能转换 javax.security.auth.PrivateCredentialPermission.implies
未能转换 javax.security.auth.PrivateCredentialPermission.newPermissionCollection
未能转换 javax.security.auth.PrivateCredentialPermission.PrivateCredentialPermission
未能转换 javax.security.auth.Refreshable
未能转换 javax.security.auth.RefreshFailedException
未能转换 javax.security.auth.RefreshFailedException.RefreshFailedException
未能转换 javax.security.auth.spi.LoginModule.abort
未能转换 javax.security.auth.spi.LoginModule.commit
未能转换 javax.security.auth.spi.LoginModule.initialize
未能转换 javax.security.auth.spi.LoginModule.login
未能转换 javax.security.auth.spi.LoginModule.logout
未能转换 javax.security.auth.Subject.doAs
未能转换 javax.security.auth.Subject.doAsPrivileged
未能转换 javax.security.auth.Subject.getPrincipals
未能转换 javax.security.auth.Subject.getPrivateCredentials
未能转换 javax.security.auth.Subject.getPublicCredentials
未能转换 javax.security.auth.Subject.getSubject
未能转换 javax.security.auth.Subject.isReadOnly
未能转换 javax.security.auth.Subject.setReadOnly
未能转换 javax.security.auth.Subject.Subject
未能转换 javax.security.auth.SubjectDomainCombiner
未能转换 javax.security.cert.Certificate.Certificate
未能转换 javax.security.cert.Certificate.getEncoded
未能转换 javax.security.cert.Certificate.verify(PublicKey)
未能转换 javax.security.cert.Certificate.verify(PublicKey, String)
未能转换 javax.security.cert.X509Certificate.getInstance
未能转换 javax.security.cert.X509Certificate.getNotAfter
未能转换 javax.security.cert.X509Certificate.getNotBefore
未能转换 javax.security.cert.X509Certificate.getSigAlgName
未能转换 javax.security.cert.X509Certificate.getVersion
未能转换 javax.security.cert.X509Certificate.X509Certificate

Javax.servlet 错误信息

未能转换 javax.servlet.FilterConfig
未能转换 javax.servlet.GenericServlet.destroy
未能转换 javax.servlet.GenericServlet.getInitParameter
未能转换 javax.servlet.GenericServlet.getInitParameterNames
未能转换 javax.servlet.GenericServlet.getServletConfig
未能转换 javax.servlet.GenericServlet.getServletContext
未能转换 javax.servlet.GenericServlet.getServletInfo
未能转换 javax.servlet.GenericServlet.getServletName
未能转换 javax.servlet.GenericServlet.init
未能转换 javax.servlet.GenericServlet.log(String)
未能转换 javax.servlet.GenericServlet.log(String, Throwable)
未能转换 javax.servlet.GenericServlet.service
未能转换 javax.servlet.HttpDummyBase.Config
未能转换 javax.servlet.HttpDummyBase.HttpDummyBase
未能转换 javax.servlet.RequestDispatcher
未能转换 javax.servlet.RequestDispatcher.forward
未能转换 javax.servlet.RequestDispatcher.include
未能转换 javax.servlet.Servlet.destroy
未能转换 javax.servlet.Servlet.getServletConfig
未能转换 javax.servlet.Servlet.getServletInfo
未能转换 javax.servlet.Servlet.init
未能转换 javax.servlet.Servlet.service
未能转换 javax.servlet.ServletConfig
未能转换 javax.servlet.ServletContext.getContext
未能转换 javax.servlet.ServletContext.getInitParameter
未能转换 javax.servlet.ServletContext.getInitParameterNames
未能转换 javax.servlet.ServletContext.getMajorVersion
未能转换 javax.servlet.ServletContext.getMimeType
未能转换 javax.servlet.ServletContext.getMinorVersion
未能转换 javax.servlet.ServletContext.getNamedDispatcher
未能转换 javax.servlet.ServletContext.getRequestDispatcher
未能转换 javax.servlet.ServletContext.getResource
未能转换 javax.servlet.ServletContext.getResourceAsStream
未能转换 javax.servlet.ServletContext.getResourcePaths
未能转换 javax.servlet.ServletContext.getServerInfo
未能转换 javax.servlet.ServletContext.getServlet
未能转换 javax.servlet.ServletContext.getServletContextName

未能转换 javax.servlet.ServletContext.getServletNames
未能转换 javax.servlet.ServletContext.getServlets
未能转换 javax.servlet.ServletContext.log
未能转换 javax.servlet.ServletContext.setAttribute
未能转换 javax.servlet.ServletContextAttributeEvent
未能转换 javax.servlet.ServletContextAttributeListener
未能转换 javax.servlet.ServletContextEvent
未能转换 javax.servlet.ServletContextListener
未能转换 javax.servlet.ServletInputStream
未能转换 javax.servlet.ServletInputStream.readLine
未能转换 javax.servlet.ServletRequest.getAttribute
未能转换 javax.servlet.ServletRequest.getAttributeNames
未能转换 javax.servlet.ServletRequest.getLocale
未能转换 javax.servlet.ServletRequest.getLocales
未能转换 javax.servlet.ServletRequest.getParameter
未能转换 javax.servlet.ServletRequest.getParameterMap
未能转换 javax.servlet.ServletRequest.getParameterNames
未能转换 javax.servlet.ServletRequest.getParameterValues
未能转换 javax.servlet.ServletRequest.getProtocol
未能转换 javax.servlet.ServletRequest.getReader
未能转换 javax.servlet.ServletRequest.getRealPath
未能转换 javax.servlet.ServletRequest.getRequestDispatcher
未能转换 javax.servlet.ServletRequest.getScheme
未能转换 javax.servlet.ServletRequest.getServerPort
未能转换 javax.servlet.ServletRequest.removeAttribute
未能转换 javax.servlet.ServletRequest.setAttribute
未能转换 javax.servlet.ServletRequest.setCharacterEncoding
未能转换 javax.servlet.ServletRequestWrapper.getAttribute
未能转换 javax.servlet.ServletRequestWrapper.getAttributeNames
未能转换 javax.servlet.ServletRequestWrapper.getLocale
未能转换 javax.servlet.ServletRequestWrapper.getLocales
未能转换 javax.servlet.ServletRequestWrapper.getParameter
未能转换 javax.servlet.ServletRequestWrapper.getParameterMap
未能转换 javax.servlet.ServletRequestWrapper.getReader
未能转换 javax.servlet.ServletRequestWrapper.getRequestDispatcher
未能转换 javax.servlet.ServletRequestWrapper.getScheme
未能转换 javax.servlet.ServletResponse
未能转换 javax.servlet.ServletResponse.flushBuffer
未能转换 javax.servlet.ServletResponse.getBufferSize
未能转换 javax.servlet.ServletResponse.getLocale

未能转换 javax.servlet.ServletResponse.isCommitted
未能转换 javax.servlet.ServletResponse.resetBuffer
未能转换 javax.servlet.ServletResponse.setBufferSize
未能转换 javax.servlet.ServletResponse.setContentLength
未能转换 javax.servlet.ServletResponseWrapper.flushBuffer
未能转换 javax.servlet.ServletResponseWrapper.getBufferSize
未能转换 javax.servlet.ServletResponseWrapper.getLocale
未能转换 javax.servlet.ServletResponseWrapper.isCommitted
未能转换 javax.servlet.ServletResponseWrapper.setBufferSize
未能转换 javax.servlet.ServletResponseWrapper.setContentLength
未能转换 javax.servlet.ServletResponseWrapper.setLocale
未能转换 javax.servlet.UnavailableException.getServlet
未能转换 javax.servlet.UnavailableException.getUnavailableSeconds
未能转换 javax.servlet.UnavailableException.isPermanent
未能转换 javax.servlet.UnavailableException.UnavailableException

Javax.servlet.http 错误信息

未能转换 javax.servlet.http.Cookie.clone
未能转换 javax.servlet.http.Cookie.getComment
未能转换 javax.servlet.http.Cookie.getMaxAge
未能转换 javax.servlet.http.Cookie.getSecure
未能转换 javax.servlet.http.Cookie.getVersion
未能转换 javax.servlet.http.Cookie.setComment
未能转换 javax.servlet.http.Cookie.setMaxAge
未能转换 javax.servlet.http.Cookie.setSecure
未能转换 javax.servlet.http.Cookie.setVersion
未能转换 javax.servlet.http.HttpServlet.getLastModified
未能转换 javax.servlet.http.HttpServlet.HttpServlet
未能转换 javax.servlet.http.HttpServlet.service(HttpServletRequest, HttpServletResponse)
未能转换 javax.servlet.http.HttpServlet.service(ServletRequest, SeervletResponse)
未能转换 javax.servlet.http.HttpServletRequest.BASIC_AUTH
未能转换 javax.servlet.http.HttpServletRequest.CLIENT_CERT_AUTH
未能转换 javax.servlet.http.HttpServletRequest.DIGEST_AUTH
未能转换 javax.servlet.http.HttpServletRequest.FORM_AUTH
未能转换 javax.servlet.http.HttpServletRequest.getQueryString
未能转换 javax.servlet.http.HttpServletRequest.getSession
未能转换 javax.servlet.http.HttpServletRequest.isRequestedSessionIdFromCookie
未能转换 javax.servlet.http.HttpServletRequest.isRequestedSessionIdFromURL
未能转换 javax.servlet.http.HttpServletRequest.isRequestedSessionIdValid
未能转换 javax.servlet.http.HttpServletRequestWrapper.getSession
未能转换 javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdFromCookie
未能转换 javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdFromURL
未能转换 javax.servlet.http.HttpServletRequestWrapper.isRequestedSessionIdValid
未能转换 javax.servlet.http.HttpServletResponse.addDateHeader
未能转换 javax.servlet.http.HttpServletResponse.containsHeader
未能转换 javax.servlet.http.HttpServletResponse.encodeRedirectURL
未能转换 javax.servlet.http.HttpServletResponse.encodeURL
未能转换 javax.servlet.http.HttpServletResponse.SC_TEMPORARY_REDIRECT
未能转换 javax.servlet.http.HttpServletResponse.setDateHeader
未能转换 javax.servlet.http.HttpServletResponseWrapper.containsHeader
未能转换 javax.servlet.http.HttpServletResponseWrapper.setDateHeader
未能转换 javax.servlet.http.HttpServletResponseWrapper.setHeader
未能转换 javax.servlet.http.HttpServletResponseWrapper.setIntHeader
未能转换 javax.servlet.http.HttpSession.getCreationTime

未能转换 javax.servlet.http.HttpSession.getLastAccessedTime
未能转换 javax.servlet.http.HttpSession.getMaxInactiveInterval
未能转换 javax.servlet.http.HttpSession.getSessionContext
未能转换 javax.servlet.http.HttpSession.getValue
未能转换 javax.servlet.http.HttpSession.invalidate
未能转换 javax.servlet.http.HttpSession.putValue
未能转换 javax.servlet.http.HttpSession.removeValue
未能转换 javax.servlet.http.HttpSession.setMaxInactiveInterval
未能转换 javax.servlet.http.HttpSessionActivationListener
未能转换 javax.servlet.http.HttpSessionAttributeListener
未能转换 javax.servlet.http.HttpSessionBindingEvent
未能转换 javax.servlet.http.HttpSessionBindingListener
未能转换 javax.servlet.http.HttpSessionContext
未能转换 javax.servlet.http.HttpSessionEvent
未能转换 javax.servlet.http.HttpSessionListener
未能转换 javax.servlet.http.HttpUtils

Javax.servlet.jsp 错误信息

未能转换 javax.servlet.jsp.HttpJspPage
未能转换 javax.servlet.jsp.JspEngineInfo
未能转换 javax.servlet.jsp.JspFactory
未能转换 javax.servlet.jsp.JspPage
未能转换 javax.servlet.jsp.JspWriter.bufferSize
未能转换 javax.servlet.jsp.JspWriter.DEFAULT_BUFFER
未能转换 javax.servlet.jsp.JspWriter.getBufferSize
未能转换 javax.servlet.jsp.JspWriter.getRemaining
未能转换 javax.servlet.jsp.JspWriter.NO_BUFFER
未能转换 javax.servlet.jsp.JspWriter.UNBOUNDED_BUFFER
未能转换 javax.servlet.jsp.PageContext.APPLICATION
未能转换 javax.servlet.jsp.PageContext.APPLICATION_SCOPE
未能转换 javax.servlet.jsp.PageContext.CONFIG
未能转换 javax.servlet.jsp.PageContext.EXCEPTION
未能转换 javax.servlet.jsp.PageContext.getAttribute
未能转换 javax.servlet.jsp.PageContext.getAttribute
未能转换 javax.servlet.jsp.PageContext.getAttributeNamesInScope
未能转换 javax.servlet.jsp.PageContext.getAttributesScope
未能转换 javax.servlet.jsp.PageContext.getServletConfig
未能转换 javax.servlet.jsp.PageContext.handlePageException
未能转换 javax.servlet.jsp.PageContext.initialize
未能转换 javax.servlet.jsp.PageContext.OUT
未能转换 javax.servlet.jsp.PageContext.PAGE
未能转换 javax.servlet.jsp.PageContext.PageContext
未能转换 javax.servlet.jsp.PageContext.PAGECONTEXT
未能转换 javax.servlet.jsp.PageContext.PAGE_SCOPE
未能转换 javax.servlet.jsp.PageContext.popBody
未能转换 javax.servlet.jsp.PageContext.pushBody
未能转换 javax.servlet.jsp.PageContext.release
未能转换 javax.servlet.jsp.PageContext.removeAttribute
未能转换 javax.servlet.jsp.PageContext.REQUEST
未能转换 javax.servlet.jsp.PageContext.REQUEST_SCOPE
未能转换 javax.servlet.jsp.PageContext.RESPONSE
未能转换 javax.servlet.jsp.PageContext.SESSION
未能转换 javax.servlet.jsp.PageContext.SESSION_SCOPE
未能转换 javax.servlet.jsp.PageContext.setAttribute
未能转换 javax.servlet.jsp.tagext.BodyContent.flush

未能转换 javax.servlet.jsp.tagext.PageData

未能转换 javax.servlet.jsp.tagext.Tag.setParent

未能转换 javax.servlet.jsp.tagext.TagAttributeInfo

未能转换 javax.servlet.jsp.tagext.TagData

未能转换 javax.servlet.jsp.tagext.TagExtraInfo

未能转换 javax.servlet.jsp.tagext.TagInfo

未能转换 javax.servlet.jsp.tagext.TagLibraryInfo

未能转换 javax.servlet.jsp.tagext.TagLibraryValidator

未能转换 javax.servlet.jsp.tagext.TagSupport.setParent

未能转换 javax.servlet.jsp.tagext.TagVariableInfo

未能转换 javax.servlet.jsp.tagext.TryCatchFinally

未能转换 javax.servlet.jsp.tagext.ValidationMessage

未能转换 javax.servlet.jsp.tagext.VariableInfo

Java.sound 错误信息

未能转换 javax.sound.midi.ControllerEventListener
未能转换 javax.sound.midi.Instrument
未能转换 javax.sound.midi.InvalidMidiDataException
未能转换 javax.sound.midi.MetaEventListener
未能转换 javax.sound.midi.MetaMessage
未能转换 javax.sound.midi.MidiChannel
未能转换 javax.sound.midi.MidiDevice
未能转换 javax.sound.midi.MidiDevice.Info
未能转换 javax.sound.midi.MidiEvent
未能转换 javax.sound.midi.MidiFormatException
未能转换 javax.sound.midi.MidiMessage
未能转换 javax.sound.midi.MidiSystem
未能转换 javax.sound.midi.MidiUnavailableException
未能转换 javax.sound.midi.Patch
未能转换 javax.sound.midi.Receiver
未能转换 javax.sound.midi.Sequence
未能转换 javax.sound.midi.Sequencer
未能转换 javax.sound.midi.Sequencer.SyncMode
未能转换 javax.sound.midi.ShortMessage
未能转换 javax.sound.midi.Soundbank
未能转换 javax.sound.midi.SoundbankResource
未能转换 javax.sound.midi.spi.MidiDeviceProvider
未能转换 javax.sound.midi.spi.MidiFileReader
未能转换 javax.sound.midi.spi.MidiFileWriter
未能转换 javax.sound.midi.spi.SoundbankReader
未能转换 javax.sound.midi.Synthesizer
未能转换 javax.sound.midi.SysexMessage
未能转换 javax.sound.midi.Track
未能转换 javax.sound.midi.Transmitter
未能转换 javax.sound.midi.VoiceStatus
未能转换 javax.sound.sampled.AudioFileFormat
未能转换 javax.sound.sampled.AudioFileFormat.Type
未能转换 javax.sound.sampled.AudioFormat.bigEndian
未能转换 javax.sound.sampled.AudioFormat.Encoding.<EncodingType>
未能转换 javax.sound.sampled.AudioFormat.Encoding.Encoding
未能转换 javax.sound.sampled.AudioFormat.isBigEndian
未能转换 javax.sound.sampled.AudioFormat.matches

未能转换 javax.sound.sampled.AudioInputStream.AudioInputStream
未能转换 javax.sound.sampled.AudioInputStream.available
未能转换 javax.sound.sampled.AudioInputStream.format
未能转换 javax.sound.sampled.AudioInputStream.frameLength
未能转换 javax.sound.sampled.AudioInputStream.framePos
未能转换 javax.sound.sampled.AudioInputStream.frameSize
未能转换 javax.sound.sampled.AudioInputStream.getFormat
未能转换 javax.sound.sampled.AudioInputStream.getFrameLength
未能转换 javax.sound.sampled.AudioInputStream.mark
未能转换 javax.sound.sampled.AudioInputStream.markSupported
未能转换 javax.sound.sampled.AudioPermission
未能转换 javax.sound.sampled.AudioSystem
未能转换 javax.sound.sampled.BooleanControl
未能转换 javax.sound.sampled.BooleanControl.Type
未能转换 javax.sound.sampled.Clip.getLength
未能转换 javax.sound.sampled.Clip.getMicrosecondLength
未能转换 javax.sound.sampled.Clip.loop
未能转换 javax.sound.sampled.Clip.open
未能转换 javax.sound.sampled.Clip.setLoopPoints
未能转换 javax.sound.sampled.Clip.setMicrosecondPosition
未能转换 javax.sound.sampled.CompoundControl
未能转换 javax.sound.sampled.CompoundControl.Type
未能转换 javax.sound.sampled.Control
未能转换 javax.sound.sampled.Control.Type
未能转换 javax.sound.sampled.DataLine.available
未能转换 javax.sound.sampled.DataLine.drain
未能转换 javax.sound.sampled.DataLine.flush
未能转换 javax.sound.sampled.DataLine.getMicrosecondPosition
未能转换 javax.sound.sampled.DataLine.Info
未能转换 javax.sound.sampled.DataLine.start
未能转换 javax.sound.sampled.EnumControl
未能转换 javax.sound.sampled.EnumControl.Type
未能转换 javax.sound.sampled.FloatControl
未能转换 javax.sound.sampled.FloatControl.Type
未能转换 javax.sound.sampled.Line
未能转换 javax.sound.sampled.Line.Info
未能转换 javax.sound.sampled.LineEvent
未能转换 javax.sound.sampled.LineEvent.Type
未能转换 javax.sound.sampled.LineListener
未能转换 javax.sound.sampled.Mixer

未能转换 javax.sound.sampled.Mixer.Info
未能转换 javax.sound.sampled.Port
未能转换 javax.sound.sampled.Port.Info
未能转换 javax.sound.sampled.ReverbType
未能转换 javax.sound.sampled.SourceDataLine
未能转换 javax.sound.sampled.SourceDataLine.open
未能转换 javax.sound.sampled.SourceDataLine.write
未能转换 javax.sound.sampled.spi.AudioFileReader
未能转换 javax.sound.sampled.spi.AudioFileWriter
未能转换 javax.sound.sampled.spi.FormatConversionProvider
未能转换 javax.sound.sampled.spi.MixerProvider
未能转换 javax.sound.sampled.TargetDataLine
未能转换 javax.sound.sampled.TargetDataLine.open
未能转换 javax.sound.sampled.TargetDataLine.read

Javax.sql 错误信息

未能转换 javax.sql.ConnectionEvent
未能转换 javax.sql.ConnectionEventListener
未能转换 javax.sql.ConnectionPoolDataSource
未能转换 javax.sql.DataSource
未能转换 javax.sql.PooledConnection.addConnectionEventListener
未能转换 javax.sql.PooledConnection.removeConnectionEventListener
未能转换 javax.sql.RowSet.addRowSetListener
未能转换 javax.sql.RowSet.getEscapeProcessing
未能转换 javax.sql.RowSet.getMaxFieldSize
未能转换 javax.sql.RowSet.getMaxRows
未能转换 javax.sql.RowSet.getPassword
未能转换 javax.sql.RowSet.getTypeMap
未能转换 javax.sql.RowSet.getUrl
未能转换 javax.sql.RowSet.getUsername
未能转换 javax.sql.RowSet.isReadOnly
未能转换 javax.sql.RowSet.removeRowSetListener
未能转换 javax.sql.RowSet.setAsciiStream
未能转换 javax.sql.RowSet.setBinaryStream
未能转换 javax.sql.RowSet.setBlob
未能转换 javax.sql.RowSet.setCharacterStream
未能转换 javax.sql.RowSet.setClob
未能转换 javax.sql.RowSet.setConcurrency
未能转换 javax.sql.RowSet.setDataSourceName
未能转换 javax.sql.RowSet.setEscapeProcessing
未能转换 javax.sql.RowSet.setMaxFieldSize
未能转换 javax.sql.RowSet.setMaxRows
未能转换 javax.sql.RowSet.setObject
未能转换 javax.sql.RowSet.setPassword
未能转换 javax.sql.RowSet.setReadOnly
未能转换 javax.sql.RowSet.setType
未能转换 javax.sql.RowSet.getTypeMap
未能转换 javax.sql.RowSet.setUrl
未能转换 javax.sql.RowSet.setUsername
未能转换 javax.sql.RowSetEvent
未能转换 javax.sql.RowSetInternal
未能转换 javax.sql.RowSetListener
未能转换 javax.sql.RowSetMetaData.setCatalogName

未能转换 javax.sql.RowSetMetaData.setColumnCount
未能转换 javax.sql.RowSetMetaData.setCurrency
未能转换 javax.sql.RowSetMetaData.setPrecision
未能转换 javax.sql.RowSetMetaData.setScale
未能转换 javax.sql.RowSetMetaData.setSearchable
未能转换 javax.sql.RowSetMetaData.setSigned
未能转换 javax.sql.RowSetReader
未能转换 javax.sql.RowSetWriter
未能转换 javax.sql.XAConnection
未能转换 javax.sql.XAConnection.getXAResource
未能转换 javax.sql.XADataSource

Javax.swing 错误信息

未能转换 javax.swing.<ClassName>.addNotify
未能转换 javax.swing.<ClassName>.process*Event
未能转换 javax.swing.<ClassName>.removeNotify
未能转换 javax.swing.<ClassName>.setLayout
未能转换 javax.swing.<ClassName>BeanInfo
未能转换 javax.swing.AbstractAction.AbstractAction(String)
未能转换 javax.swing.AbstractAction.AbstractAction(String, Icon)
未能转换 javax.swing.AbstractAction.addPropertyChangeListener
未能转换 javax.swing.AbstractAction.changeSupport
未能转换 javax.swing.AbstractAction.enabled
未能转换 javax.swing.AbstractAction.firePropertyChange
未能转换 javax.swing.AbstractAction.getKeys
未能转换 javax.swing.AbstractAction.getValue
未能转换 javax.swing.AbstractAction.isEnabled
未能转换 javax.swing.AbstractAction.putValue
未能转换 javax.swing.AbstractAction.removePropertyChangeListener
未能转换 javax.swing.AbstractAction.setEnabled
未能转换 javax.swing.AbstractButton.<ChangedProperty>
未能转换 javax.swing.AbstractButton.AbstractButton
未能转换 javax.swing.AbstractButton.actionListener
未能转换 javax.swing.AbstractButton.addChangeListener
未能转换 javax.swing.AbstractButton.changeEvent
未能转换 javax.swing.AbstractButton.changeListener
未能转换 javax.swing.AbstractButton.checkHorizontalKey
未能转换 javax.swing.AbstractButton.checkVerticalKey
未能转换 javax.swing.AbstractButton.configurePropertiesFromAction
未能转换 javax.swing.AbstractButton.createActionListener
未能转换 javax.swing.AbstractButton.createActionPropertyChangeListener
未能转换 javax.swing.AbstractButton.createChangeListener
未能转换 javax.swing.AbstractButton.createItemListener
未能转换 javax.swing.AbstractButton.doClick
未能转换 javax.swing.AbstractButton.fireItemStateChanged
未能转换 javax.swing.AbstractButton.fireStateChanged
未能转换 javax.swing.AbstractButton.get<State>Icon
未能转换 javax.swing.AbstractButton.getAction
未能转换 javax.swing.AbstractButton.getActionCommand
未能转换 javax.swing.AbstractButton.getHorizontalAlignment

未能转换 javax.swing.AbstractButton.getHorizontalTextPosition
未能转换 javax.swing.AbstractButton.getIcon
未能转换 javax.swing.AbstractButton.getMargin
未能转换 javax.swing.AbstractButton.getMnemonic
未能转换 javax.swing.AbstractButton.getModel
未能转换 javax.swing.AbstractButton.getSelectedObjects
未能转换 javax.swing.AbstractButton.getUI
未能转换 javax.swing.AbstractButton.getVerticalAlignment
未能转换 javax.swing.AbstractButton.getVerticalTextPosition
未能转换 javax.swing.AbstractButton.imageUpdate
未能转换 javax.swing.AbstractButton.init
未能转换 javax.swing.AbstractButton.isBorderPainted
未能转换 javax.swing.AbstractButton.isContentAreaFilled
未能转换 javax.swing.AbstractButton.isFocusPainted
未能转换 javax.swing.AbstractButton.isRolloverEnabled
未能转换 javax.swing.AbstractButton.itemListener
未能转换 javax.swing.AbstractButton.model
未能转换 javax.swing.AbstractButton.paintBorder
未能转换 javax.swing.AbstractButton.removeChangeListener
未能转换 javax.swing.AbstractButton.set<State>Icon
未能转换 javax.swing.AbstractButton.setActionCommand
未能转换 javax.swing.AbstractButton.setBorderPainted
未能转换 javax.swing.AbstractButton.setContentAreaFilled
未能转换 javax.swing.AbstractButton.setFocusPainted
未能转换 javax.swing.AbstractButton.setHorizontalTextPosition
未能转换 javax.swing.AbstractButton.setHorizontalTextPosition
未能转换 javax.swing.AbstractButton.setIcon
未能转换 javax.swing.AbstractButton.setMargin
未能转换 javax.swing.AbstractButton.setMnemonic
未能转换 javax.swing.AbstractButton.setModel
未能转换 javax.swing.AbstractButton.setRolloverEnabled
未能转换 javax.swing.AbstractButton.setSelected
未能转换 javax.swing.AbstractButton.setUI
未能转换 javax.swing.AbstractButton.setVerticalAlignment
未能转换 javax.swing.AbstractButton.setVerticalTextPosition
未能转换 javax.swing.AbstractButton.updateUI
未能转换 javax.swing.AbstractCellEditor
未能转换 javax.swing.AbstractListModel
未能转换 javax.swing.AbstractListModel.addListDataListener
未能转换 javax.swing.AbstractListModel.fireContentsChanged

未能转换 javax.swing.AbstractListModel.fireIntervalAdded
未能转换 javax.swing.AbstractListModel.fireIntervalRemoved
未能转换 javax.swing.AbstractListModel.getListeners
未能转换 javax.swing.AbstractListModel.listenerList
未能转换 javax.swing.AbstractListModel.removeListDataListener
未能转换 javax.swing.Action.ACCELERATOR_KEY
未能转换 javax.swing.Action.ACTION_COMMAND_KEY
未能转换 javax.swing.Action.addPropertyChangeListener
未能转换 javax.swing.Action.DEFAULT
未能转换 javax.swing.Action.getValue
未能转换 javax.swing.Action.isEnabled
未能转换 javax.swing.Action.LONG_DESCRIPTION
未能转换 javax.swing.Action.MNEMONIC_KEY
未能转换 javax.swing.Action.NAME
未能转换 javax.swing.Action.putValue
未能转换 javax.swing.Action.removePropertyChangeListener
未能转换 javax.swing.Action.setEnabled
未能转换 javax.swing.Action.SHORT_DESCRIPTION
未能转换 javax.swing.Action.SMALL_ICON
未能转换 javax.swing.ActionMap
未能转换 javax.swing.ActionMap.allKeys
未能转换 javax.swing.ActionMap.get
未能转换 javax.swing.ActionMap.getParent
未能转换 javax.swing.ActionMap.keys
未能转换 javax.swing.ActionMap.setParent
未能转换 javax.swing.ActionMap.size
未能转换 javax.swing.BorderFactory
未能转换 javax.swing.BorderFactory.BorderFactory
未能转换 javax.swing.BorderFactory.createCompoundBorder
未能转换 javax.swing.BorderFactory.createEtchedBorder
未能转换 javax.swing.BorderFactory.createMatteBorder
未能转换 javax.swing.BorderFactory.createTitledBorder
未能转换 javax.swing.BoundedRangeModel
未能转换 javax.swing.Box
未能转换 javax.swing.Box.Filler
未能转换 javax.swing.BoxLayout
未能转换 javax.swing.ButtonGroup
未能转换 javax.swing.ButtonGroup.add
未能转换 javax.swing.ButtonModel
未能转换 javax.swing.ButtonModel.addActionListener

未能转换 javax.swing.ButtonModel.addChangeListener
未能转换 javax.swing.ButtonModel.addItemListener
未能转换 javax.swing.ButtonModel.getActionCommand
未能转换 javax.swing.ButtonModel.getMnemonic
未能转换 javax.swing.ButtonModel.isArmed
未能转换 javax.swing.ButtonModel.isPressed
未能转换 javax.swing.ButtonModel.isRollover
未能转换 javax.swing.ButtonModel.isSelected
未能转换 javax.swing.ButtonModel.removeActionListener
未能转换 javax.swing.ButtonModel.removeChangeListener
未能转换 javax.swing.ButtonModel.removeItemListener
未能转换 javax.swing.ButtonModel.setActionCommand
未能转换 javax.swing.ButtonModel.setArmed
未能转换 javax.swing.ButtonModel.setGroup
未能转换 javax.swing.ButtonModel.setMnemonic
未能转换 javax.swing.ButtonModel.setPressed
未能转换 javax.swing.ButtonModel.setRollover
未能转换 javax.swing.ButtonModel.setSelected
未能转换 javax.swing.CellEditor
未能转换 javax.swing.CellRendererPane
未能转换 javax.swing.ComboBoxEditor
未能转换 javax.swing.ComboBoxModel.getSelectedItem
未能转换 javax.swing.ComboBoxModel.setSelectedItem
未能转换 javax.swing.ComponentInputMap
未能转换 javax.swing.DebugGraphics
未能转换 javax.swing.DebugGraphics.BUFFERED_OPTION
未能转换 javax.swing.DebugGraphics.clearRect
未能转换 javax.swing.DebugGraphics.clipRect
未能转换 javax.swing.DebugGraphics.copyArea
未能转换 javax.swing.DebugGraphics.create
未能转换 javax.swing.DebugGraphics.DebugGraphics
未能转换 javax.swing.DebugGraphics.DebugGraphics(Graphics)
未能转换 javax.swing.DebugGraphics.DebugGraphics (Graphics, IComponent)
未能转换 javax.swing.DebugGraphics.dispose
未能转换 javax.swing.DebugGraphics.draw3DRect
未能转换 javax.swing.DebugGraphics.drawArc
未能转换 javax.swing.DebugGraphics.drawBytes
未能转换 javax.swing.DebugGraphics.drawChars
未能转换 javax.swing.DebugGraphics.drawImage
未能转换 javax.swing.DebugGraphics.drawLine

未能转换 javax.swing.DebugGraphics.drawOval
未能转换 javax.swing.DebugGraphics.drawPolygon
未能转换 javax.swing.DebugGraphics.drawPolyline
未能转换 javax.swing.DebugGraphics.drawRect
未能转换 javax.swing.DebugGraphics.drawRoundRect
未能转换 javax.swing.DebugGraphics.drawString(AttributedCharacterIterator, int, int)
未能转换 javax.swing.DebugGraphics.drawString(String, int, int)
未能转换 javax.swing.DebugGraphics.fill3DRect
未能转换 javax.swing.DebugGraphics.fillArc
未能转换 javax.swing.DebugGraphics.fillOval
未能转换 javax.swing.DebugGraphics.fillPolygon
未能转换 javax.swing.DebugGraphics.fillRect
未能转换 javax.swing.DebugGraphics.fillRect
未能转换 javax.swing.DebugGraphics.FLASH_OPTION
未能转换 javax.swing.DebugGraphics.flashColor
未能转换 javax.swing.DebugGraphics.flashCount
未能转换 javax.swing.DebugGraphics.flashTime
未能转换 javax.swing.DebugGraphics.getClip
未能转换 javax.swing.DebugGraphics.getClipBounds
未能转换 javax.swing.DebugGraphics.getColor
未能转换 javax.swing.DebugGraphics.getDebugOptions
未能转换 javax.swing.DebugGraphics.getFont
未能转换 javax.swing.DebugGraphics.getFontMetrics
未能转换 javax.swing.DebugGraphics.isDrawingBuffer
未能转换 javax.swing.DebugGraphics.LOG_OPTION
未能转换 javax.swing.DebugGraphics.logStream
未能转换 javax.swing.DebugGraphics.NONE_OPTION
未能转换 javax.swing.DebugGraphics.setClip
未能转换 javax.swing.DebugGraphics.setColor
未能转换 javax.swing.DebugGraphics.setDebugOptions
未能转换 javax.swing.DebugGraphics.setFlashColor
未能转换 javax.swing.DebugGraphics.setFlashCount
未能转换 javax.swing.DebugGraphics.setFlashTime
未能转换 javax.swing.DebugGraphics.setFont
未能转换 javax.swing.DebugGraphics.setLogStream
未能转换 javax.swing.DebugGraphics.setPaintMode
未能转换 javax.swing.DebugGraphics.setXORMode
未能转换 javax.swing.DebugGraphics.translate
未能转换 javax.swing.DefaultBoundedRangeModel
未能转换 javax.swing.DefaultButtonModel

未能转换 javax.swing.DefaultButtonModel.actionCommand
未能转换 javax.swing.DefaultButtonModel.addActionListener
未能转换 javax.swing.DefaultButtonModel.addChangeListener
未能转换 javax.swing.DefaultButtonModel.addItemListener
未能转换 javax.swing.DefaultButtonModel.ARMED
未能转换 javax.swing.DefaultButtonModel.changeEvent
未能转换 javax.swing.DefaultButtonModel.DefaultButtonModel
未能转换 javax.swing.DefaultButtonModel.ENABLED
未能转换 javax.swing.DefaultButtonModel.fireActionPerformed
未能转换 javax.swing.DefaultButtonModel.fireItemStateChanged
未能转换 javax.swing.DefaultButtonModel.fireStateChanged
未能转换 javax.swing.DefaultButtonModel.getActionCommand
未能转换 javax.swing.DefaultButtonModel.getGroup
未能转换 javax.swing.DefaultButtonModel.getListeners
未能转换 javax.swing.DefaultButtonModel.getMnemonic
未能转换 javax.swing.DefaultButtonModel.getSelectedObjects
未能转换 javax.swing.DefaultButtonModel.group
未能转换 javax.swing.DefaultButtonModel.isArmed
未能转换 javax.swing.DefaultButtonModel.isPressed
未能转换 javax.swing.DefaultButtonModel.isRollover
未能转换 javax.swing.DefaultButtonModel.isSelected
未能转换 javax.swing.DefaultButtonModel.listenerList
未能转换 javax.swing.DefaultButtonModel.mnemonic
未能转换 javax.swing.DefaultButtonModel.PRESSED
未能转换 javax.swing.DefaultButtonModel.removeActionListener
未能转换 javax.swing.DefaultButtonModel.removeChangeListener
未能转换 javax.swing.DefaultButtonModel.removeItemListener
未能转换 javax.swing.DefaultButtonModel.ROLLOVER
未能转换 javax.swing.DefaultButtonModel.SELECTED
未能转换 javax.swing.DefaultButtonModel.setActionCommand
未能转换 javax.swing.DefaultButtonModel.setArmed
未能转换 javax.swing.DefaultButtonModel.setGroup
未能转换 javax.swing.DefaultButtonModel.setMnemonic
未能转换 javax.swing.DefaultButtonModel.setPressed
未能转换 javax.swing.DefaultButtonModel.setRollover
未能转换 javax.swing.DefaultButtonModel.setSelected
未能转换 javax.swing.DefaultButtonModel.stateMask
未能转换 javax.swing.DefaultCellEditor
未能转换 javax.swing.DefaultComboBoxModel
未能转换 javax.swing.DefaultComboBoxModel.DefaultComboBoxModel

未能转换 javax.swing.DefaultComboBoxModel.DefaultComboBoxModel(Object[])
未能转换 javax.swing.DefaultComboBoxModel.DefaultComboBoxModel(Vector)
未能转换 javax.swing.DefaultComboBoxModel.getSelectedItem
未能转换 javax.swing.DefaultComboBoxModel.setSelectedItem
未能转换 javax.swing.DefaultDesktopManager
未能转换 javax.swing.DefaultDesktopManager.deiconifyFrame
未能转换 javax.swing.DefaultDesktopManager.iconifyFrame
未能转换 javax.swing.DefaultDesktopManager.maximizeFrame
未能转换 javax.swing.DefaultDesktopManager.minimizeFrame
未能转换 javax.swing.DefaultFocusManager
未能转换 javax.swing.DefaultListCellRenderer
未能转换 javax.swing.DefaultListCellRenderer.UIResource
未能转换 javax.swing.DefaultListModel
未能转换 javax.swing.DefaultListModel.capacity
未能转换 javax.swing.DefaultListModel.ensureCapacity
未能转换 javax.swing.DefaultListModel.setSize
未能转换 javax.swing.DefaultListModel.trimToSize
未能转换 javax.swing.DefaultListSelectionModel
未能转换 javax.swing.DefaultListSelectionModel.addListSelectionListener
未能转换 javax.swing.DefaultListSelectionModel.addSelectionInterval
未能转换 javax.swing.DefaultListSelectionModel.clone
未能转换 javax.swing.DefaultListSelectionModel.DefaultListSelectionModel
未能转换 javax.swing.DefaultListSelectionModel.fireValueChanged(boolean)
未能转换 javax.swing.DefaultListSelectionModel.fireValueChanged(int, int)
未能转换 javax.swing.DefaultListSelectionModel.fireValueChanged(int, int, boolean)
未能转换 javax.swing.DefaultListSelectionModel.getAnchorSelectionIndex
未能转换 javax.swing.DefaultListSelectionModel.getLeadSelectionIndex
未能转换 javax.swing.DefaultListSelectionModel.getListeners
未能转换 javax.swing.DefaultListSelectionModel.getMaxSelectionIndex
未能转换 javax.swing.DefaultListSelectionModel.getMinSelectionIndex
未能转换 javax.swing.DefaultListSelectionModel.getSelectionMode
未能转换 javax.swing.DefaultListSelectionModel.getValuesAdjusting
未能转换 javax.swing.DefaultListSelectionModel.insertIndexInterval
未能转换 javax.swing.DefaultListSelectionModel.isLeadAnchorNotificationEnabled
未能转换 javax.swing.DefaultListSelectionModel.leadAnchorNotificationEnabled
未能转换 javax.swing.DefaultListSelectionModel.listenerList
未能转换 javax.swing.DefaultListSelectionModel.removeIndexInterval
未能转换 javax.swing.DefaultListSelectionModel.removeListSelectionListener
未能转换 javax.swing.DefaultListSelectionModel.removeSelectionInterval
未能转换 javax.swing.DefaultListSelectionModel.setAnchorSelectionIndex

未能转换 javax.swing.DefaultListSelectionModel.setLeadAnchorNotificationEnabled
未能转换 javax.swing.DefaultListSelectionModel.setLeadSelectionIndex
未能转换 javax.swing.DefaultListSelectionModel.setSelectionInterval
未能转换 javax.swing.DefaultListSelectionModel.setSelectionMode
未能转换 javax.swing.DefaultListSelectionModel.setValuesAdjusting
未能转换 javax.swing.DefaultSingleSelectionModel
未能转换 javax.swing.DefaultSingleSelectionModel.addChangeListener
未能转换 javax.swing.DefaultSingleSelectionModel.changeEvent
未能转换 javax.swing.DefaultSingleSelectionModel.DefaultSingleSelectionModel
未能转换 javax.swing.DefaultSingleSelectionModel.fireStateChanged
未能转换 javax.swing.DefaultSingleSelectionModel.getListeners
未能转换 javax.swing.DefaultSingleSelectionModel.listenerList
未能转换 javax.swing.DefaultSingleSelectionModel.removeChangeListener
未能转换 javax.swing.DesktopManager
未能转换 javax.swing.DesktopManager.deiconifyFrame
未能转换 javax.swing.DesktopManager.iconifyFrame
未能转换 javax.swing.DesktopManager.maximizeFrame
未能转换 javax.swing.DesktopManager.minimizeFrame
未能转换 javax.swing.FocusManager
未能转换 javax.swing.GrayFilter
未能转换 javax.swing.GrayFilter.createDisabledImage
未能转换 javax.swing.Icon.paintIcon
未能转换 javax.swing.ImageIcon
未能转换 javax.swing.ImageIcon.component
未能转换 javax.swing.ImageIcon.getAccessibleContext
未能转换 javax.swing.ImageIcon.getDescription
未能转换 javax.swing.ImageIcon.getImageLoadStatus
未能转换 javax.swing.ImageIcon.getImageObserver
未能转换 javax.swing.ImageIcon.ImageIcon
未能转换 javax.swing.ImageIcon.ImageIcon(byte[])
未能转换 javax.swing.ImageIcon.ImageIcon(byte[], String)
未能转换 javax.swing.ImageIcon.ImageIcon(Image)
未能转换 javax.swing.ImageIcon.ImageIcon(Image, String)
未能转换 javax.swing.ImageIcon.ImageIcon(String)
未能转换 javax.swing.ImageIcon.ImageIcon(String, String)
未能转换 javax.swing.ImageIcon.ImageIcon(URL)
未能转换 javax.swing.ImageIcon.ImageIcon(URL, String)
未能转换 javax.swing.ImageIcon.loadImage
未能转换 javax.swing.ImageIcon.paintIcon
未能转换 javax.swing.ImageIcon.setDescription

未能转换 javax.swing.ImageIcon.setImageObserver
未能转换 javax.swing.ImageIcon.tracker
未能转换 javax.swing.InputMap
未能转换 javax.swing.InputMap.allKeys
未能转换 javax.swing.InputMap.get
未能转换 javax.swing.InputMap.getParent
未能转换 javax.swing.InputMap.keys
未能转换 javax.swing.InputMap.setParent
未能转换 javax.swing.InputVerifier
未能转换 javax.swing.JApplet
未能转换 javax.swing.JApplet.accessibleContext
未能转换 javax.swing.JApplet.addImpl
未能转换 javax.swing.JApplet.createRootPane
未能转换 javax.swing.JApplet.getGlassPane
未能转换 javax.swing.JApplet.getJMenuBar
未能转换 javax.swing.JApplet.getLayeredPane
未能转换 javax.swing.JApplet.getRootPane
未能转换 javax.swing.JApplet.isRootPaneCheckingEnabled
未能转换 javax.swing.JApplet paramString
未能转换 javax.swing.JApplet.rootPane
未能转换 javax.swing.JApplet.rootPaneCheckingEnabled
未能转换 javax.swing.JApplet.setContentPane
未能转换 javax.swing.JApplet.setGlassPane
未能转换 javax.swing.JApplet.setJMenuBar
未能转换 javax.swing.JApplet.setLayeredPane
未能转换 javax.swing.JApplet.setRootPane
未能转换 javax.swing.JApplet.setRootPaneCheckingEnabled
未能转换 javax.swing.JButton.configurePropertiesFromAction
未能转换 javax.swing.JButton.isDefaultButton
未能转换 javax.swing.JButton.isDefaultCapable
未能转换 javax.swing.JButton.JButton
未能转换 javax.swing.JButton paramString
未能转换 javax.swing.JButton.setDefaultCapable
未能转换 javax.swing.JButton.updateUI
未能转换 javax.swing.JCheckBox.configurePropertiesFromAction
未能转换 javax.swing.JCheckBox.createActionPropertyChangeListener
未能转换 javax.swing.JCheckBox.JCheckBox
未能转换 javax.swing.JCheckBox.updateUI
未能转换 javax.swing.JCheckBoxMenuItem.getAccessibleContext
未能转换 javax.swing.JCheckBoxMenuItem.getUIClassID

未能转换 javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(Action)
未能转换 javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(Icon)
未能转换 javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, boolean)
未能转换 javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, Icon)
未能转换 javax.swing.JCheckBoxMenuItem.JCheckBoxMenuItem(String, Icon, boolean)
未能转换 javax.swing.JCheckBoxMenuItem.requestFocus
未能转换 javax.swing.JColorChooser.accessibleContext
未能转换 javax.swing.JColorChooser.addChooserPanel
未能转换 javax.swing.JColorChooser.CHOOSER_PANELS_PROPERTY
未能转换 javax.swing.JColorChooser.createDialog
未能转换 javax.swing.JColorChooser.getAccessibleContext
未能转换 javax.swing.JColorChooser.getChooserPanels
未能转换 javax.swing.JColorChooser.getPreviewPanel
未能转换 javax.swing.JColorChooser.getSelectionModel
未能转换 javax.swing.JColorChooser.getUI
未能转换 javax.swing.JColorChooser.getUIClassID
未能转换 javax.swing.JColorChooser.JColorChooser
未能转换 javax.swing.JColorChooser.PREVIEW_PANEL_PROPERTY
未能转换 javax.swing.JColorChooser.removeChooserPanel
未能转换 javax.swing.JColorChooser.SELECTION_MODEL_PROPERTY
未能转换 javax.swing.JColorChooser.setChooserPanels
未能转换 javax.swing.JColorChooser.setPreviewPanel
未能转换 javax.swing.JColorChooser.setSelectionModel
未能转换 javax.swing.JColorChooser.setUI
未能转换 javax.swing.JColorChooser.showDialog
未能转换 javax.swing.JColorChooser.updateUI
未能转换 javax.swing.JComboBox.actionCommand
未能转换 javax.swing.JComboBox.actionPerformed
未能转换 javax.swing.JComboBox.configureEditor
未能转换 javax.swing.JComboBox.configurePropertiesFromAction
未能转换 javax.swing.JComboBox.contentsChanged
未能转换 javax.swing.JComboBox.createActionPropertyChangeListener
未能转换 javax.swing.JComboBox.createDefaultKeySelectionManager
未能转换 javax.swing.JComboBox.editor
未能转换 javax.swing.JComboBox.getAction
未能转换 javax.swing.JComboBox.getActionCommand
未能转换 javax.swing.JComboBox.getEditor
未能转换 javax.swing.JComboBox.getKeySelectionManager
未能转换 javax.swing.JComboBox.getRenderer
未能转换 javax.swing.JComboBox.getUI

未能转换 javax.swing.JComboBox.getUIClassID
未能转换 javax.swing.JComboBox.installAncestorListener
未能转换 javax.swing.JComboBox.intervalAdded
未能转换 javax.swing.JComboBox.intervalRemoved
未能转换 javax.swing.JComboBox.isLightWeightPopupEnabled
未能转换 javax.swing.JComboBox.JComboBox
未能转换 javax.swing.JComboBox.keySelectionManager
未能转换 javax.swing.JComboBox.KeySelectionManager
未能转换 javax.swing.JComboBox.lightWeightPopupEnabled
未能转换 javax.swing.JComboBox.renderer
未能转换 javax.swing.JComboBox.selectedItemReminder
未能转换 javax.swing.JComboBox.selectWithKeyChar
未能转换 javax.swing.JComboBox.setActionCommand
未能转换 javax.swing.JComboBox.setEditor
未能转换 javax.swing.JComboBox.setKeySelectionManager
未能转换 javax.swing.JComboBox.setLightWeightPopupEnabled
未能转换 javax.swing.JComboBox.setModel
未能转换 javax.swing.JComboBox.setRenderer
未能转换 javax.swing.JComboBox.setUI
未能转换 javax.swing.JComboBox.updateUI
未能转换 javax.swing.JComponent.accessibleContext
未能转换 javax.swing.JComponent.AccessibleJComponent
未能转换 javax.swing.JComponent.addAncestorListener
未能转换 javax.swing.JComponent.addNotify
未能转换 javax.swing.JComponent.addPropertyChangeListener
未能转换 javax.swing.JComponent.addVetoableChangeListener
未能转换 javax.swing.JComponent.computeVisibleRect
未能转换 javax.swing.JComponent.createToolTip
未能转换 javax.swing.JComponent.disable
未能转换 javax.swing.JComponent.firePropertyChange
未能转换 javax.swing.JComponent.fireVetoableChange
未能转换 javax.swing.JComponent.getActionForKeyStroke
未能转换 javax.swing.JComponent.getActionMap
未能转换 javax.swing.JComponent.getAlignmentX
未能转换 javax.swing.JComponent.getAlignmentY
未能转换 javax.swing.JComponent.getAutoscrolls
未能转换 javax.swing.JComponent.getBorder
未能转换 javax.swing.JComponent.getClientProperty
未能转换 javax.swing.JComponent.getConditionForKeyStroke
未能转换 javax.swing.JComponent.getDebugGraphicsOptions

未能转换 javax.swing.JComponent.getInputMap
未能转换 javax.swing.JComponent getInputVerifier
未能转换 javax.swing.JComponent getListeners
未能转换 javax.swing.JComponent getLocation
未能转换 javax.swing.JComponent getMaximumSize
未能转换 javax.swing.JComponent getMinimumSize
未能转换 javax.swing.JComponent getNextFocusableComponent
未能转换 javax.swing.JComponent getPreferredSize
未能转换 javax.swing.JComponent getRegisteredKeyStrokes
未能转换 javax.swing.JComponent getRootPane
未能转换 javax.swing.JComponent getSize
未能转换 javax.swing.JComponent getToolTipLocation
未能转换 javax.swing.JComponent getToolTipText
未能转换 javax.swing.JComponent getTopLevelAncestor
未能转换 javax.swing.JComponent verifyInputWhenFocusTarget
未能转换 javax.swing.JComponent getVisibleRect
未能转换 javax.swing.JComponent isDoubleBuffered
未能转换 javax.swing.JComponent isFocusCycleRoot
未能转换 javax.swing.JComponent isLightweightComponent
未能转换 javax.swing.JComponent isManagingFocus
未能转换 javax.swing.JComponent isMaximumSizeSet
未能转换 javax.swing.JComponent isMinimumSizeSet
未能转换 javax.swing.JComponent isOpaque
未能转换 javax.swing.JComponent isOptimizedDrawingEnabled
未能转换 javax.swing.JComponent isPaintingTile
未能转换 javax.swing.JComponent isPreferredSizeSet
未能转换 javax.swing.JComponent isValidateRoot
未能转换 javax.swing.JComponent listenerList
未能转换 javax.swing.JComponent paint
未能转换 javax.swing.JComponent paintBorder
未能转换 javax.swing.JComponent paintChildren
未能转换 javax.swing.JComponent paintComponent
未能转换 javax.swing.JComponent paintImmediately
未能转换 javax.swing.JComponent print
未能转换 javax.swing.JComponent printAll
未能转换 javax.swing.JComponent printBorder
未能转换 javax.swing.JComponent printChildren
未能转换 javax.swing.JComponent printComponent
未能转换 javax.swing.JComponent processKeyBinding
未能转换 javax.swing.JComponent putClientProperty

未能转换 javax.swing.JComponent.registerKeyboardAction
未能转换 javax.swing.JComponent.removeAncestorListener
未能转换 javax.swing.JComponent.removePropertyChangeListener
未能转换 javax.swing.JComponent.removeVetoableChangeListener
未能转换 javax.swing.JComponent.repaint
未能转换 javax.swing.JComponent.requestDefaultFocus
未能转换 javax.swing.JComponent.requestFocus
未能转换 javax.swing.JComponent.resetKeyboardActions
未能转换 javax.swing.JComponent.scrollRectToVisible
未能转换 javax.swing.JComponent.setActionMap
未能转换 javax.swing.JComponent.setAlignmentX
未能转换 javax.swing.JComponent.setAlignmentY
未能转换 javax.swing.JComponent.setAutoscrolls
未能转换 javax.swing.JComponent.setBorder
未能转换 javax.swing.JComponent.setDebugGraphicsOptions
未能转换 javax.swing.JComponent.setDoubleBuffered
未能转换 javax.swing.JComponent.setEnabled
未能转换 javax.swing.JComponent.setInputMap
未能转换 javax.swing.JComponent.setInputVerifier
未能转换 javax.swing.JComponent.setMaximumSize
未能转换 javax.swing.JComponent.setMinimumSize
未能转换 javax.swing.JComponent.setNextFocusableComponent
未能转换 javax.swing.JComponent.setOpaque
未能转换 javax.swing.JComponent.setPreferredSize
未能转换 javax.swing.JComponent.setRequestFocusEnabled
未能转换 javax.swing.JComponent.setToolTipText
未能转换 javax.swing.JComponent.setUI
未能转换 javax.swing.JComponent.verifyInputWhenFocusTarget
未能转换 javax.swing.JComponent.setVisible
未能转换 javax.swing.JComponent.TOOT_TIP_TEXT_KEY
未能转换 javax.swing.JComponent.ui
未能转换 javax.swing.JComponent.unregisterKeyboardAction
未能转换 javax.swing.JComponent.updateUI
未能转换 javax.swing.JDesktopPane
未能转换 javax.swing.JDesktopPane.<DragMode>
未能转换 javax.swing.JDesktopPane.getDesktopManager
未能转换 javax.swing.JDesktopPane.getDragMode
未能转换 javax.swing.JDesktopPane.getUI
未能转换 javax.swing.JDesktopPane.getUIClassID
未能转换 javax.swing.JDesktopPane.isOpaque

未能转换 javax.swing.JDesktopPane.JDesktopPane
未能转换 javax.swing.JDesktopPane.setDesktopManager
未能转换 javax.swing.JDesktopPane.setDragMode
未能转换 javax.swing.JDesktopPane.setSelectedFrame
未能转换 javax.swing.JDesktopPane.setUI
未能转换 javax.swing.JDesktopPane.updateUI
未能转换 javax.swing.JDialog.accessibleContext
未能转换 javax.swing.JDialog.addImpl
未能转换 javax.swing.JDialog.dialogInit
未能转换 javax.swing.JDialog.getDefaultCloseOperation
未能转换 javax.swing.JDialog.getGlassPane
未能转换 javax.swing.JDialog.getLayeredPane
未能转换 javax.swing.JDialog.getRootPane
未能转换 javax.swing.JDialog.isRootPaneCheckingEnabled
未能转换 javax.swing.JDialog.JDialog
未能转换 javax.swing.JDialog.JDialog(Dialog, boolean)
未能转换 javax.swing.JDialog.JDialog(Dialog, String, boolean)
未能转换 javax.swing.JDialog.JDialog(Frame, boolean)
未能转换 javax.swing.JDialog.JDialog(Frame, String, boolean)
未能转换 javax.swing.JDialog.rootPane
未能转换 javax.swing.JDialog.rootPaneCheckingEnabled
未能转换 javax.swing.JDialog.setContentPane
未能转换 javax.swing.JDialog.setDefaultCloseOperation
未能转换 javax.swing.JDialog.setGlassPane
未能转换 javax.swing.JDialog.setLayeredPane
未能转换 javax.swing.JDialog.setLocationRelativeTo
未能转换 javax.swing.JDialog.setRootPane
未能转换 javax.swing.JDialog.setRootPaneCheckingEnabled
未能转换 javax.swing.JEditorPane
未能转换 javax.swing.JEditorPane.createDefaultEditorKit
未能转换 javax.swing.JEditorPane.createEditorKitForContentType
未能转换 javax.swing.JEditorPane.fireHyperlinkUpdate
未能转换 javax.swing.JEditorPane.getContentType
未能转换 javax.swing.JEditorPane.getEditorKit
未能转换 javax.swing.JEditorPane.getEditorKitClassNameForContentType
未能转换 javax.swing.JEditorPane.getEditorKitForContentType
未能转换 javax.swing.JEditorPane.getPage
未能转换 javax.swing.JEditorPane.getPreferredSize
未能转换 javax.swing.JEditorPane.getScrollableTracksViewportHeight
未能转换 javax.swing.JEditorPane.getScrollableTracksViewportWidth

未能转换 javax.swing.JEditorPane.getStream
未能转换 javax.swing.JEditorPane.getUIClassID
未能转换 javax.swing.JEditorPane.isFocusCycleRoot
未能转换 javax.swing.JEditorPane.JEditorPane(String)
未能转换 javax.swing.JEditorPane.JEditorPane(String, String)
未能转换 javax.swing.JEditorPane.JEditorPane(URL)
未能转换 javax.swing.JEditorPane.read
未能转换 javax.swing.JEditorPane.registerEditorKitForContentType(String, String)
未能转换 javax.swing.JEditorPane.registerEditorKitForContentType(String, String, ClassLoader)
未能转换 javax.swing.JEditorPane.scrollToReference
未能转换 javax.swing.JEditorPane.setContentType
未能转换 javax.swing.JEditorPane.setEditorKit
未能转换 javax.swing.JEditorPane.setEditorKitForContentType
未能转换 javax.swing.JEditorPane setPage
未能转换 javax.swing.JFileChooser.accept
未能转换 javax.swing.JFileChooser.accessibleContext
未能转换 javax.swing.JFileChooser.addChoosableFileFilter
未能转换 javax.swing.JFileChooser.approveSelection
未能转换 javax.swing.JFileChooser.cancelSelection
未能转换 javax.swing.JFileChooser.changeToParentDirectory
未能转换 javax.swing.JFileChooser.ensureFileIsVisible
未能转换 javax.swing.JFileChooser.fireActionPerformed
未能转换 javax.swing.JFileChooser.getAcceptAllFileFilter
未能转换 javax.swing.JFileChooser.getAccessibleContext
未能转换 javax.swing.JFileChooser.getAccessory
未能转换 javax.swing.JFileChooser.getApproveButtonMnemonic
未能转换 javax.swing.JFileChooser.getApproveButtonText
未能转换 javax.swing.JFileChooser.getApproveButtonToolTipText
未能转换 javax.swing.JFileChooser.getChoosableFileFilters
未能转换 javax.swing.JFileChooser.getCurrentDirectory
未能转换 javax.swing.JFileChooser.getDescription
未能转换 javax.swing.JFileChooser.getFileFilter
未能转换 javax.swing.JFileChooser.getFileSelectionMode
未能转换 javax.swing.JFileChooser.getFileSystemView
未能转换 javax.swing.JFileChooser.getFileView
未能转换 javax.swing.JFileChooser.getIcon
未能转换 javax.swing.JFileChooser.getSelectedFiles
未能转换 javax.swing.JFileChooser.getTypeDescription
未能转换 javax.swing.JFileChooser.getUI
未能转换 javax.swing.JFileChooser.isAcceptAllFileFilterUsed

未能转换 javax.swing.JFileChooser.isFileHidingEnabled
未能转换 javax.swing.JFileChooser.isFileSelectionEnabled
未能转换 javax.swing.JFileChooser.isTraversable
未能转换 javax.swing.JFileChooser.JFileChooser(File, FileSystemView)
未能转换 javax.swing.JFileChooser.JFileChooser(FileSystemView)
未能转换 javax.swing.JFileChooser.JFileChooser(String, FileSystemView)
未能转换 javax.swing.JFileChooser.removeChoosableFileFilter
未能转换 javax.swing.JFileChooser.rescanCurrentDirectory
未能转换 javax.swing.JFileChooser.resetChoosableFileFilters
未能转换 javax.swing.JFileChooser.setAcceptAllFileFilterUsed
未能转换 javax.swing.JFileChooser.setAccessory
未能转换 javax.swing.JFileChooser.setApproveButtonMnemonic
未能转换 javax.swing.JFileChooser.setApproveButtonText
未能转换 javax.swing.JFileChooser.setApproveButtonToolTipText
未能转换 javax.swing.JFileChooser.setControlButtonsAreShown
未能转换 javax.swing.JFileChooser.setCurrentDirectory
未能转换 javax.swing.JFileChooser.setDialogType
未能转换 javax.swing.JFileChooser.setFileFilter
未能转换 javax.swing.JFileChooser.setFileHidingEnabled
未能转换 javax.swing.JFileChooser.setFileSelectionMode
未能转换 javax.swing.JFileChooser.setFileSystemView
未能转换 javax.swing.JFileChooser.setFileView
未能转换 javax.swing.JFileChooser.setSelectedFiles
未能转换 javax.swing.JFileChooser.setup
未能转换 javax.swing.JFileChooser.showDialog
未能转换 javax.swing.JFileChooser.showSaveDialog
未能转换 javax.swing.JFileChooser.updateUI
未能转换 javax.swing.JFrame.EXIT_ON_CLOSE
未能转换 javax.swing.JFrame.frameInit
未能转换 javax.swing.JFrame.getContentPane
未能转换 javax.swing.JFrame.getDefaultCloseOperation
未能转换 javax.swing.JFrame.getGlassPane
未能转换 javax.swing.JFrame.getLayeredPane
未能转换 javax.swing.JFrame.isRootPaneCheckingEnabled
未能转换 javax.swing.JFrame.rootPane
未能转换 javax.swing.JFrame.rootPaneCheckingEnabled
未能转换 javax.swing.JFrame.setContentPane
未能转换 javax.swing.JFrame.setDefaultCloseOperation
未能转换 javax.swing.JFrame.setGlassPane
未能转换 javax.swing.JFrame.setLayeredPane

未能转换 javax.swing.JFrame.setRootPane
未能转换 javax.swing.JFrame.setRootPaneCheckingEnabled
未能转换 javax.swing.JInternalFrame
未能转换 javax.swing.JInternalFrame.<PropertyName>
未能转换 javax.swing.JInternalFrame.closable
未能转换 javax.swing.JInternalFrame.desktopIcon
未能转换 javax.swing.JInternalFrame.doDefaultCloseAction
未能转换 javax.swing.JInternalFrame.fireInternalFrameEvent
未能转换 javax.swing.JInternalFrame.getDefaultCloseOperation
未能转换 javax.swing.JInternalFrame.getDesktopIcon
未能转换 javax.swing.JInternalFrame.getGlassPane
未能转换 javax.swing.JInternalFrame.getLayer
未能转换 javax.swing.JInternalFrame.getLayeredPane
未能转换 javax.swing.JInternalFrame.getNormalBounds
未能转换 javax.swing.JInternalFrame.getUI
未能转换 javax.swing.JInternalFrame.getUIClassID
未能转换 javax.swing.JInternalFrame.iconable
未能转换 javax.swing.JInternalFrame.isClosable
未能转换 javax.swing.JInternalFrame.isClosed
未能转换 javax.swing.JInternalFrame.isIcon
未能转换 javax.swing.JInternalFrame.isMaximum
未能转换 javax.swing.JInternalFrame.isRootPaneCheckingEnabled
未能转换 javax.swing.JInternalFrame.isSelected
未能转换 javax.swing.JInternalFrame.JDesktopIcon
未能转换 javax.swing.JInternalFrame.maximizable
未能转换 javax.swing.JInternalFrame.moveToBack
未能转换 javax.swing.JInternalFrame.moveToFront
未能转换 javax.swing.JInternalFrame.pack
未能转换 javax.swing.JInternalFrame.paintComponent
未能转换 javax.swing.JInternalFrame paramString
未能转换 javax.swing.JInternalFrame.resizable
未能转换 javax.swing.JInternalFrame.restoreSubcomponentFocus
未能转换 javax.swing.JInternalFrame.rootPane
未能转换 javax.swing.JInternalFrame.rootPaneCheckingEnabled
未能转换 javax.swing.JInternalFrame.setClosable
未能转换 javax.swing.JInternalFrame.setClosed
未能转换 javax.swing.JInternalFrame.setContentPane
未能转换 javax.swing.JInternalFrame.setDefaultCloseOperation
未能转换 javax.swing.JInternalFrame.setDesktopIcon
未能转换 javax.swing.JInternalFrame.setFrameIcon

未能转换 javax.swing.JInternalFrame.setGlassPane
未能转换 javax.swing.JInternalFrame.setLayer
未能转换 javax.swing.JInternalFrame.setLayeredPane
未能转换 javax.swing.JInternalFrame.setNormalBounds
未能转换 javax.swing.JInternalFrame.setRootPane
未能转换 javax.swing.JInternalFrame.setRootPaneCheckingEnabled
未能转换 javax.swing.JInternalFrame.setSelected
未能转换 javax.swing.JInternalFrame.setUI
未能转换 javax.swing.JInternalFrame.updateUI
未能转换 javax.swing.JLabel.checkHorizontalKey
未能转换 javax.swing.JLabel.checkVerticalKey
未能转换 javax.swing.JLabel.getDisabledIcon
未能转换 javax.swing.JLabel.getDisplayedMnemonic
未能转换 javax.swing.JLabel.getIconTextGap
未能转换 javax.swing.JLabel.getLabelFor
未能转换 javax.swing.JLabel.getUI
未能转换 javax.swing.JLabel.getUIClassID
未能转换 javax.swing.JLabel.imageUpdate
未能转换 javax.swing.JLabel.labelXFor
未能转换 javax.swing.JLabel.setDisabledIcon
未能转换 javax.swing.JLabel.setHorizontalAlignment
未能转换 javax.swing.JLabel.setHorizontalTextPosition
未能转换 javax.swing.JLabel.setIconTextGap
未能转换 javax.swing.JLabel.setLabelFor
未能转换 javax.swing.JLabel.setUI
未能转换 javax.swing.JLabel.setVerticalAlignment
未能转换 javax.swing.JLabel.setVerticalTextPosition
未能转换 javax.swing.JLabel.updateUI
未能转换 javax.swing.JLayeredPane
未能转换 javax.swing.JLayeredPane.addImpl
未能转换 javax.swing.JLayeredPane.getComponentCountInLayer
未能转换 javax.swing.JLayeredPane.getComponentsInLayer
未能转换 javax.swing.JLayeredPane.getComponentToLayer
未能转换 javax.swing.JLayeredPane.indexOf
未能转换 javax.swing.JLayeredPane.getLayer
未能转换 javax.swing.JLayeredPane.getLayeredPaneAbove
未能转换 javax.swing.JLayeredPane.getObjectForLayer
未能转换 javax.swing.JLayeredPane.getPosition
未能转换 javax.swing.JLayeredPane.highestLayer
未能转换 javax.swing.JLayeredPane.insertIndexForLayer

未能转换 javax.swing.JLayeredPane.isOptimizedDrawingEnabled
未能转换 javax.swing.JLayeredPane.JLayeredPane
未能转换 javax.swing.JLayeredPane.LAYER_PROPERTY
未能转换 javax.swing.JLayeredPane.lowestLayer
未能转换 javax.swing.JLayeredPane.moveToBack
未能转换 javax.swing.JLayeredPane.moveToFront
未能转换 javax.swing.JLayeredPane.paramString
未能转换 javax.swing.JLayeredPane.putLayer
未能转换 javax.swing.JLayeredPane.remove
未能转换 javax.swing.JLayeredPane.setLayer
未能转换 javax.swing.JLayeredPane.setPosition
未能转换 javax.swing.JList.addListSelectionListener
未能转换 javax.swing.JList.addSelectionInterval
未能转换 javax.swing.JList.createSelectionModel
未能转换 javax.swing.JList.ensureIndexIsVisible
未能转换 javax.swing.JList.fireSelectionValueChanged
未能转换 javax.swing.JList.getAnchorSelectionIndex
未能转换 javax.swing.JList.getCellBounds
未能转换 javax.swing.JList.getCellRenderer
未能转换 javax.swing.JList.getFixedCellHeight
未能转换 javax.swing.JList.getFixedCellWidth
未能转换 javax.swing.JList.getLastVisibleIndex
未能转换 javax.swing.JList.getLeadSelectionIndex
未能转换 javax.swing.JList.getMaxSelectionIndex
未能转换 javax.swing.JList.getMinSelectionIndex
未能转换 javax.swing.JList.getModel
未能转换 javax.swing.JList.getPreferredSize
未能转换 javax.swing.JList.getPrototypeCellValue
未能转换 javax.swing.JList.getScrollableBlockIncrement
未能转换 javax.swing.JList.getScrollableTracksViewportHeight
未能转换 javax.swing.JList.getScrollableTracksViewportWidth
未能转换 javax.swing.JList.getScrollableUnitIncrement
未能转换 javax.swing.JList.setSelectedIndex
未能转换 javax.swing.JList.getSelectedIndices
未能转换 javax.swing.JList.getSelectedValues
未能转换 javax.swing.JList.getSelectionBackground
未能转换 javax.swing.JList.getSelectionForeground
未能转换 javax.swing.JList.getSelectionModel
未能转换 javax.swing.JList.getUI
未能转换 javax.swing.JList.getUIClassID

未能转换 javax.swing.JList.getValueIsAdjusting
未能转换 javax.swing.JList.getVisibleRowCount
未能转换 javax.swing.JList.indexToLocation
未能转换 javax.swing.JList.isSelectedIndex
未能转换 javax.swing.JList.JList(ListModel)
未能转换 javax.swing.JList.JList(Object[])
未能转换 javax.swing.JList paramString
未能转换 javax.swing.JList.removeListSelectionListener
未能转换 javax.swing.JList.removeSelectionInterval
未能转换 javax.swing.JList.setCellRenderer
未能转换 javax.swing.JList.setFixedCellHeight
未能转换 javax.swing.JList.setFixedCellWidth
未能转换 javax.swing.JList.setListData
未能转换 javax.swing.JList.setModel
未能转换 javax.swing.JList.setPrototypeCellValue
未能转换 javax.swing.JList.setSelectedIndices
未能转换 javax.swing.JList.setSelectionBackground
未能转换 javax.swing.JList.setSelectionForeground
未能转换 javax.swing.JList.setSelectionInterval
未能转换 javax.swing.JList.setSelectionModel
未能转换 javax.swing.JList.setUI
未能转换 javax.swing.JList.setValueIsAdjusting
未能转换 javax.swing.JList.setVisibleRowCount
未能转换 javax.swing.JList.updateUI
未能转换 javax.swing.JMenu.add(Component)
未能转换 javax.swing.JMenu.add(Component, int)
未能转换 javax.swing.JMenu.add(MenuItem)
未能转换 javax.swing.JMenu.createActionChangeListener
未能转换 javax.swing.JMenu.createWinListener
未能转换 javax.swing.JMenu.doClick
未能转换 javax.swing.JMenu.fireMenuCanceled
未能转换 javax.swing.JMenu.fireMenuDeselected
未能转换 javax.swing.JMenu.fireMenuItemSelected
未能转换 javax.swing.JMenu.getAccessibleContext
未能转换 javax.swing.JMenu.getComponent
未能转换 javax.swing.JMenu.getDelay
未能转换 javax.swing.JMenu getMenuComponent
未能转换 javax.swing.JMenu getMenuComponents
未能转换 javax.swing.JMenu.getPopupMenu
未能转换 javax.swing.JMenu.getPopupMenuOrigin

未能转换 javax.swing.JMenu.getUIClassID
未能转换 javax.swing.JMenu.isMenuItemComponent
未能转换 javax.swing.JMenu.isPopupMenuVisible
未能转换 javax.swing.JMenu.isTearOff
未能转换 javax.swing.JMenu.isTopLevelMenu
未能转换 javax.swing.JMenu.menuSelectionChanged
未能转换 javax.swing.JMenu.popupListener
未能转换 javax.swing.JMenu.setAccelerator
未能转换 javax.swing.JMenu.setDelay
未能转换 javax.swing.JMenu.setMenuLocation
未能转换 javax.swing.JMenu.setModel
未能转换 javax.swing.JMenu.setPopupMenuVisible
未能转换 javax.swing.JMenu.updateUI
未能转换 javax.swing.JMenuBar.getAccessibleContext
未能转换 javax.swing.JMenuBar.getComponent
未能转换 javax.swing.JMenuBar.getComponentAtIndex
未能转换 javax.swing.JMenuBar.getComponentIndex
未能转换 javax.swing.JMenuBar.getHelpMenu
未能转换 javax.swing.JMenuBar.getMargin
未能转换 javax.swing.JMenuBar.getSelectionModel
未能转换 javax.swing.JMenuBar.getUI
未能转换 javax.swing.JMenuBar.getUIClassID
未能转换 javax.swing.JMenuBar.isBorderPainted
未能转换 javax.swing.JMenuBar.isManagingFocus
未能转换 javax.swing.JMenuBar.isSelected
未能转换 javax.swing.JMenuBar.menuSelectionChanged
未能转换 javax.swing.JMenuBar.paintBorder
未能转换 javax.swing.JMenuBar.processKeyBinding
未能转换 javax.swing.JMenuBar.setBorderPainted
未能转换 javax.swing.JMenuBar.setHelpMenu
未能转换 javax.swing.JMenuBar.setMargin
未能转换 javax.swing.JMenuBar.setSelected
未能转换 javax.swing.JMenuBar.setSelectionModel
未能转换 javax.swing.JMenuBar.setUI
未能转换 javax.swing.JMenuBar.updateUI
未能转换 javax.swing.JMenuItem.addMenuDragMouseListener
未能转换 javax.swing.JMenuItem.addMenuKeyListener
未能转换 javax.swing.JMenuItem.configurePropertiesFromAction
未能转换 javax.swing.JMenuItem.createActionPropertyChangeListener
未能转换 javax.swing.JMenuItem.fireMenuDragMouseDragged

未能转换 javax.swing.JMenuItem.fireMenuDragMouseEntered
未能转换 javax.swing.JMenuItem.fireMenuDragMouseExited
未能转换 javax.swing.JMenuItem.fireMenuDragMouseReleased
未能转换 javax.swing.JMenuItem.fireMenuKeyPressed
未能转换 javax.swing.JMenuItem.fireMenuKeyReleased
未能转换 javax.swing.JMenuItem.fireMenuKeyTyped
未能转换 javax.swing.JMenuItem.getAccelerator
未能转换 javax.swing.JMenuItem.getAccessibleContext
未能转换 javax.swing.JMenuItem.getComponent
未能转换 javax.swing.JMenuItem.getUIClassID
未能转换 javax.swing.JMenuItem.init
未能转换 javax.swing.JMenuItem.isArmed
未能转换 javax.swing.JMenuItem.JMenuItem
未能转换 javax.swing.JMenuItem.menuSelectionChanged
未能转换 javax.swing.JMenuItem.removeMenuDragMouseListener
未能转换 javax.swing.JMenuItem.removeMenuKeyListener
未能转换 javax.swing.JMenuItem.setAccelerator
未能转换 javax.swing.JMenuItem.setArmed
未能转换 javax.swing.JMenuItem.setUI
未能转换 javax.swing.JMenuItem.updateUI
未能转换 javax.swing.JOptionPane
未能转换 javax.swing.JOptionPane.showConfirmDialog
未能转换 javax.swing.JOptionPane.showInternalConfirmDialog
未能转换 javax.swing.JOptionPane.showInternalMessageDialog
未能转换 javax.swing.JOptionPane.showMessageDialog
未能转换 javax.swing.JPanel.JPanel
未能转换 javax.swing.JPasswordField.getUIClassID
未能转换 javax.swing.JPasswordField.JPasswordField
未能转换 javax.swing.JPasswordField.JPasswordField(Document, String, int)
未能转换 javax.swing.JPasswordField.JPasswordField(int)
未能转换 javax.swing.JPasswordField.JPasswordField(String, int)
未能转换 javax.swing.JPopupMenu.add
未能转换 javax.swing.JPopupMenu.createActionChangeListener
未能转换 javax.swing.JPopupMenu.createActionComponent
未能转换 javax.swing.JPopupMenu.firePopupMenuCanceled
未能转换 javax.swing.JPopupMenu.firePopupMenuWillBecomeInvisible
未能转换 javax.swing.JPopupMenu.firePopupMenuWillBecomeVisible
未能转换 javax.swing.JPopupMenu.getAccessibleContext
未能转换 javax.swing.JPopupMenu.getComponent
未能转换 javax.swing.JPopupMenu.getComponentAtIndex

未能转换 javax.swing.JPopupMenu.getComponentIndex
未能转换 javax.swing.JPopupMenu.getDefaultLightWeightPopupEnabled
未能转换 javax.swing.JPopupMenu.getInvoker
未能转换 javax.swing.JPopupMenu.getLabel
未能转换 javax.swing.JPopupMenu.getSelectionModel
未能转换 javax.swing.JPopupMenu.getUI
未能转换 javax.swing.JPopupMenu.getUIClassID
未能转换 javax.swing.JPopupMenu.insert
未能转换 javax.swing.JPopupMenu.isBorderPainted
未能转换 javax.swing.JPopupMenu.isLightWeightPopupEnabled
未能转换 javax.swing.JPopupMenu.isPopupTrigger
未能转换 javax.swing.JPopupMenu.isVisible
未能转换 javax.swing.JPopupMenu.menuSelectionChanged
未能转换 javax.swing.JPopupMenu.pack
未能转换 javax.swing.JPopupMenu.paintBorder
未能转换 javax.swing.JPopupMenu.Separator
未能转换 javax.swing.JPopupMenu.setBorderPainted
未能转换 javax.swing.JPopupMenu.setDefaultLightWeightPopupEnabled
未能转换 javax.swing.JPopupMenu.setLabel
未能转换 javax.swing.JPopupMenu.setLightWeightPopupEnabled
未能转换 javax.swing.JPopupMenu.setLocation
未能转换 javax.swing.JPopupMenu.setPopupSize(Dimension)
未能转换 javax.swing.JPopupMenu.setPopupSize(int, int)
未能转换 javax.swing.JPopupMenu.setSelected
未能转换 javax.swing.JPopupMenu.setSelectionModel
未能转换 javax.swing.JPopupMenu.setUI
未能转换 javax.swing.JPopupMenu.setVisible
未能转换 javax.swing.JPopupMenu.show
未能转换 javax.swing.JPopupMenu.updateUI
未能转换 javax.swing.JProgressBar.addChangeListener
未能转换 javax.swing.JProgressBar.changeEvent
未能转换 javax.swing.JProgressBar.changeListener
未能转换 javax.swing.JProgressBar.createChangeListener
未能转换 javax.swing.JProgressBar.fireStateChanged
未能转换 javax.swing.JProgressBar.getModel
未能转换 javax.swing.JProgressBar.getOrientation
未能转换 javax.swing.JProgressBar.getString
未能转换 javax.swing.JProgressBar.getUI
未能转换 javax.swing.JProgressBar.getUIClassID
未能转换 javax.swing.JProgressBar.isBorderPainted

未能转换 javax.swing.JProgressBar.drawStringPainted
未能转换 javax.swing.JProgressBar.JProgressBar(Bounded RangeModel)
未能转换 javax.swing.JProgressBar.JProgressBar(int)
未能转换 javax.swing.JProgressBar.JProgressBar(int, int, int)
未能转换 javax.swing.JProgressBar.model
未能转换 javax.swing.JProgressBar.orientation
未能转换 javax.swing.JProgressBar.paintBorder
未能转换 javax.swing.JProgressBar.paintString
未能转换 javax.swing.JProgressBar.progressString
未能转换 javax.swing.JProgressBar.removeChangeListener
未能转换 javax.swing.JProgressBar.setBorderPainted
未能转换 javax.swing.JProgressBar.setModel
未能转换 javax.swing.JProgressBar.setOrientation
未能转换 javax.swing.JProgressBar.setString
未能转换 javax.swing.JProgressBar.setStringPainted
未能转换 javax.swing.JProgressBar.setUI
未能转换 javax.swing.JRadioButton.configurePropertiesFromAction
未能转换 javax.swing.JRadioButton.createActionPropertyChangeListener
未能转换 javax.swing.JRadioButton.getUIClassID
未能转换 javax.swing.JRadioButton.JButton
未能转换 javax.swing.JRadioButton.updateUI
未能转换 javax.swing.JRadioButtonMenuItem.getAccessibleContext
未能转换 javax.swing.JRadioButtonMenuItem.getUIClassID
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(Action)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(Icon)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(Icon, boolean)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(String)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(String, boolean)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(String, Icon)
未能转换 javax.swing.JRadioButtonMenuItem.JButtonMenuItem(String, Icon, boolean)
未能转换 javax.swing.JRadioButtonMenuItem.requestFocus
未能转换 javax.swing.JRootPane.contentPane
未能转换 javax.swing.JRootPane.createGlassPane
未能转换 javax.swing.JRootPane.createLayeredPane
未能转换 javax.swing.JRootPane.createRootLayout
未能转换 javax.swing.JRootPane.defaultButton
未能转换 javax.swing.JRootPane.defaultPressAction
未能转换 javax.swing.JRootPane.defaultReleaseAction
未能转换 javax.swing.JRootPane.getDefaultButton

未能转换 javax.swing.JRootPane.getGlassPane
未能转换 javax.swing.JRootPane.getLayeredPane
未能转换 javax.swing.JRootPane.getUI
未能转换 javax.swing.JRootPane.getUIClassID
未能转换 javax.swing.JRootPane.glassPane
未能转换 javax.swing.JRootPane.isFocusCycleRoot
未能转换 javax.swing.JRootPane.isOptimizedDrawingEnabled
未能转换 javax.swing.JRootPane.isValidateRoot
未能转换 javax.swing.JRootPane.layeredPane
未能转换 javax.swing.JRootPane.menuBar
未能转换 javax.swing.JRootPane.setDefaultButton
未能转换 javax.swing.JRootPane.setGlassPane
未能转换 javax.swing.JRootPane.setLayeredPane
未能转换 javax.swing.JRootPane.setUI
未能转换 javax.swing.JRootPane.updateUI
未能转换 javax.swing.JScrollBar
未能转换 javax.swing.JScrollBar.fireAdjustmentValueChanged
未能转换 javax.swing.JScrollBar.getBlockIncrement
未能转换 javax.swing.JScrollBar.getMaximumSize
未能转换 javax.swing.JScrollBar.getModel
未能转换 javax.swing.JScrollBar.getUI
未能转换 javax.swing.JScrollBar.getUIClassID
未能转换 javax.swing.JScrollBar.getUnitIncrement
未能转换 javax.swing.JScrollBar.getValuesAdjusting
未能转换 javax.swing.JScrollBar.JScrollBar
未能转换 javax.swing.JScrollBar.model
未能转换 javax.swing.JScrollBar.orientation
未能转换 javax.swing.JScrollBar.setBlockIncrement
未能转换 javax.swing.JScrollBar.setModel
未能转换 javax.swing.JScrollBar.setOrientation
未能转换 javax.swing.JScrollBar.setUnitIncrement
未能转换 javax.swing.JScrollBar.setValuesAdjusting
未能转换 javax.swing.JScrollBar.setValues
未能转换 javax.swing.JScrollBar.setVisibleAmount
未能转换 javax.swing.JScrollPane
未能转换 javax.swing.JScrollPane.columnHeader
未能转换 javax.swing.JScrollPane.createHorizontalScrollBar
未能转换 javax.swing.JScrollPane.createVerticalScrollBar
未能转换 javax.swing.JScrollPane.createViewport
未能转换 javax.swing.JScrollPane.getColumnHeader

未能转换 javax.swing.JScrollPane.getCorner
未能转换 javax.swing.JScrollPane.getHorizontalScrollBar
未能转换 javax.swing.JScrollPane.getHorizontalScrollBarPolicy
未能转换 javax.swing.JScrollPane.getRowHeader
未能转换 javax.swing.JScrollPane.getUI
未能转换 javax.swing.JScrollPane.getUIClassID
未能转换 javax.swing.JScrollPane.getVerticalScrollBar
未能转换 javax.swing.JScrollPane.getVerticalScrollBarPolicy
未能转换 javax.swing.JScrollPane.setViewport
未能转换 javax.swing.JScrollPane.setViewportBorder
未能转换 javax.swing.JScrollPane.horizontalScrollBar
未能转换 javax.swing.JScrollPane.horizontalScrollBarPolicy
未能转换 javax.swing.JScrollPane.isValidateRoot
未能转换 javax.swing.JScrollPane.JScrollPane
未能转换 javax.swing.JScrollPane.JScrollPane(Component)
未能转换 javax.swing.JScrollPane.JScrollPane(Component, int, int)
未能转换 javax.swing.JScrollPane.JScrollPane(int, int)
未能转换 javax.swing.JScrollPane.lowerLeft
未能转换 javax.swing.JScrollPane.lowerRight
未能转换 javax.swing.JScrollPane.rowHeader
未能转换 javax.swing.JScrollPane.setColumnHeader
未能转换 javax.swing.JScrollPane.setCorner
未能转换 javax.swing.JScrollPane.setHorizontalScrollBar
未能转换 javax.swing.JScrollPane.setHorizontalScrollBarPolicy
未能转换 javax.swing.JScrollPane.setRowHeader
未能转换 javax.swing.JScrollPane.setUI
未能转换 javax.swing.JScrollPane.setVerticalScrollBar
未能转换 javax.swing.JScrollPane.setVerticalScrollBarPolicy
未能转换 javax.swing.JScrollPane.setViewport
未能转换 javax.swing.JScrollPane.setViewportBorder
未能转换 javax.swing.JScrollPane.updateUI
未能转换 javax.swing.JScrollPane.upperLeft
未能转换 javax.swing.JScrollPane.upperRight
未能转换 javax.swing.JScrollPane.verticalScrollBar
未能转换 javax.swing.JScrollPane.verticalScrollBarPolicy
未能转换 javax.swing.JScrollPane.viewport
未能转换 javax.swing.JSeparator
未能转换 javax.swing.JSlider.changeEvent
未能转换 javax.swing.JSlider.changeListener
未能转换 javax.swing.JSlider.createChangeListener

未能转换 javax.swing.JSlider.createStandardLabels(int)
未能转换 javax.swing.JSlider.createStandardLabels(int, int)
未能转换 javax.swing.JSlider.fireStateChanged
未能转换 javax.swing.JSlider.getExtent
未能转换 javax.swing.JSlider.getLabelTable
未能转换 javax.swing.JSlider.getMajorTickSpacing
未能转换 javax.swing.JSlider.getModel
未能转换 javax.swing.JSlider.getUI
未能转换 javax.swing.JSlider.getUIClassID
未能转换 javax.swing.JSlider.getValueIsAdjusting
未能转换 javax.swing.JSlider.JSlider
未能转换 javax.swing.JSlider.majorTickSpacing
未能转换 javax.swing.JSlider.setExtent
未能转换 javax.swing.JSlider.setInverted
未能转换 javax.swing.JSlider.setLabelTable
未能转换 javax.swing.JSlider.setMajorTickSpacing
未能转换 javax.swing.JSlider.setModel
未能转换 javax.swing.JSlider.setPaintLabels
未能转换 javax.swing.JSlider.setPaintTrack
未能转换 javax.swing.JSlider.setSnapToTicks
未能转换 javax.swing.JSlider.setUI
未能转换 javax.swing.JSlider.setValueIsAdjusting
未能转换 javax.swing.JSlider.sliderModel
未能转换 javax.swing.JSlider.snapToTicks
未能转换 javax.swing.JSlider.updateLabelUIs
未能转换 javax.swing.JSlider.updateUI
未能转换 javax.swing.JSplitPane
未能转换 javax.swing.JSplitPane.continuousLayout
未能转换 javax.swing.JSplitPane.getResizeWeight
未能转换 javax.swing.JSplitPane.getUI
未能转换 javax.swing.JSplitPane.getUIClassID
未能转换 javax.swing.JSplitPane.isContinuousLayout
未能转换 javax.swing.JSplitPane.isOneTouchExpandable
未能转换 javax.swing.JSplitPane.isValidRoot
未能转换 javax.swing.JSplitPaneJSplitPane(int, boolean)
未能转换 javax.swing.JSplitPaneJSplitPane(int, boolean, Component, Component)
未能转换 javax.swing.JSplitPane.oneTouchExpandable
未能转换 javax.swing.JSplitPane.paintChildren
未能转换 javax.swing.JSplitPane.resetToPreferredSizes
未能转换 javax.swing.JSplitPane.setContinuousLayout

未能转换 javax.swing.JSplitPane.setOneTouchExpandable
未能转换 javax.swing.JSplitPane.setResizeWeight
未能转换 javax.swing.JSplitPane.setUI
未能转换 javax.swing.JSplitPane.updateUI
未能转换 javax.swing.JTabbedPane.add(Component)
未能转换 javax.swing.JTabbedPane.add(Component, int)
未能转换 javax.swing.JTabbedPane.add(Component, Object)
未能转换 javax.swing.JTabbedPane.add(Component, Object, int)
未能转换 javax.swing.JTabbedPane.add(String, Component)
未能转换 javax.swing.JTabbedPane.addTab(String, Component)
未能转换 javax.swing.JTabbedPane.addTab(String, Icon, Component, String)
未能转换 javax.swing.JTabbedPane.changeEvent
未能转换 javax.swing.JTabbedPane.changeListener
未能转换 javax.swing.JTabbedPane.createChangeListener
未能转换 javax.swing.JTabbedPane.fireStateChanged
未能转换 javax.swing.JTabbedPane.getBackgroundAt
未能转换 javax.swing.JTabbedPane.getBoundsAt
未能转换 javax.swing.JTabbedPane.getDisabledIconAt
未能转换 javax.swing.JTabbedPane.getForegroundAt
未能转换 javax.swing.JTabbedPane.getIconAt
未能转换 javax.swing.JTabbedPane.getModel
未能转换 javax.swing.JTabbedPane.getTabPlacement
未能转换 javax.swing.JTabbedPane.getTabRunCount
未能转换 javax.swing.JTabbedPane.getToolTipText
未能转换 javax.swing.JTabbedPane.getUI
未能转换 javax.swing.JTabbedPane.getUIClassID
未能转换 javax.swing.JTabbedPane.indexOfComponent
未能转换 javax.swing.JTabbedPane.indexOfTab(Icon)
未能转换 javax.swing.JTabbedPane.indexOfTab(String)
未能转换 javax.swing.JTabbedPane.insertTab
未能转换 javax.swing.JTabbedPane.JTabbedPane
未能转换 javax.swing.JTabbedPane.model
未能转换 javax.swing.JTabbedPane.paramString
未能转换 javax.swing.JTabbedPane.remove
未能转换 javax.swing.JTabbedPane.setBackgroundAt
未能转换 javax.swing.JTabbedPane.setComponentAt
未能转换 javax.swing.JTabbedPane.setDisabledIconAt
未能转换 javax.swing.JTabbedPane.setForegroundAt
未能转换 javax.swing.JTabbedPane.setIconAt
未能转换 javax.swing.JTabbedPane.setModel

未能转换 javax.swing.JTabbedPane.setSelectedComponent
未能转换 javax.swing.JTabbedPane.setSelectedIndex
未能转换 javax.swing.JTabbedPane.setTabPlacement
未能转换 javax.swing.JTabbedPane.setToolTipTextAt
未能转换 javax.swing.JTabbedPane.setUI
未能转换 javax.swing.JTabbedPane.tabPlacement
未能转换 javax.swing.JTabbedPane.updateUI
未能转换 javax.swing.JTable
未能转换 javax.swing.JTable.<ResizeType>
未能转换 javax.swing.JTable.addColumnSelectionInterval
未能转换 javax.swing.JTable.addRowSelectionInterval
未能转换 javax.swing.JTable.autoCreateColumnsFromModel
未能转换 javax.swing.JTable.autoResizeMode
未能转换 javax.swing.JTable.cellEditor
未能转换 javax.swing.JTable.cellSelectionEnabled
未能转换 javax.swing.JTable.changeSelection
未能转换 javax.swing.JTable.clearSelection
未能转换 javax.swing.JTable.columnAdded
未能转换 javax.swing.JTable.columnAtPoint
未能转换 javax.swing.JTable.columnMarginChanged
未能转换 javax.swing.JTable.columnMoved
未能转换 javax.swing.JTable.columnRemoved
未能转换 javax.swing.JTable.columnSelectionChanged
未能转换 javax.swing.JTable.configureEnclosingScrollPane
未能转换 javax.swing.JTable.convertColumnIndexToModel
未能转换 javax.swing.JTable.convertColumnIndexToView
未能转换 javax.swing.JTable.createDefaultColumnModel
未能转换 javax.swing.JTable.createDefaultColumnsFromModel
未能转换 javax.swing.JTable.createDefaultEditors
未能转换 javax.swing.JTable.createDefaultRenderers
未能转换 javax.swing.JTable.createDefaultSelectionModel
未能转换 javax.swing.JTable.createDefaultTableHeader
未能转换 javax.swing.JTable.createScrollPaneForTable
未能转换 javax.swing.JTable.defaultEditorsByColumnClass
未能转换 javax.swing.JTable.defaultRenderersByColumnClass
未能转换 javax.swing.JTable.doLayout
未能转换 javax.swing.JTable.editCellAt
未能转换 javax.swing.JTable.editingCanceled
未能转换 javax.swing.JTable.editingColumn
未能转换 javax.swing.JTable.editingRow

未能转换 javax.swing.JTable.editingStopped
未能转换 javax.swing.JTable.editorComp
未能转换 javax.swing.JTable.getAutoCreateColumnsFromModel
未能转换 javax.swing.JTable.getAutoResizeMode
未能转换 javax.swing.JTable.getCellEditor
未能转换 javax.swing.JTable.getCellRect
未能转换 javax.swing.JTable.getCellRenderer
未能转换 javax.swing.JTable.getCellSelectionEnabled
未能转换 javax.swing.JTable.getColumn
未能转换 javax.swing.JTable getColumnSelectionAllowed
未能转换 javax.swing.JTable getDefaultEditor
未能转换 javax.swing.JTable getDefaultRenderer
未能转换 javax.swing.JTable.getEditingColumn
未能转换 javax.swing.JTable.getEditingRow
未能转换 javax.swing.JTable.getEditorComponent
未能转换 javax.swing.JTable.getIntercellSpacing
未能转换 javax.swing.JTable.getPreferredScrollableViewportSize
未能转换 javax.swing.JTable.getRowHeight
未能转换 javax.swing.JTable.getRowHeight(int)
未能转换 javax.swing.JTable.getRowMargin
未能转换 javax.swing.JTable.getRowSelectionAllowed
未能转换 javax.swing.JTable.getScrollableBlockIncrement
未能转换 javax.swing.JTable.getScrollableTracksViewportHeight
未能转换 javax.swing.JTable.getScrollableTracksViewportWidth
未能转换 javax.swing.JTable.getScrollableUnitIncrement
未能转换 javax.swing.JTable.getSelectedColumn
未能转换 javax.swing.JTable.getSelectedColumnCount
未能转换 javax.swing.JTable.getSelectedColumns
未能转换 javax.swing.JTable.getSelectedRowCount
未能转换 javax.swing.JTable.getSelectedRows
未能转换 javax.swing.JTable.getSelectionBackground
未能转换 javax.swing.JTable.getSelectionForeground
未能转换 javax.swing.JTable.getSelectionModel
未能转换 javax.swing.JTable.getShowHorizontalLines
未能转换 javax.swing.JTable.getShowVerticalLines
未能转换 javax.swing.JTable.getTableHeader
未能转换 javax.swing.JTable.getToolTipText
未能转换 javax.swing.JTable.getUI
未能转换 javax.swing.JTable.getUIClassID
未能转换 javax.swing.JTable.gridColor

未能转换 javax.swing.JTable.initializeLocalVars
未能转换 javax.swing.JTable.isCellEditable
未能转换 javax.swing.JTable.isCellSelected
未能转换 javax.swing.JTable.isColumnSelected
未能转换 javax.swing.JTable.isEditing
未能转换 javax.swing.JTable.isFocusTraversable
未能转换 javax.swing.JTable.isManagingFocus
未能转换 javax.swing.JTable.isRowSelected
未能转换 javax.swing.JTable.JTable
未能转换 javax.swing.JTable.JTable(int, int)
未能转换 javax.swing.JTable.JTable(Object[][], Object[][])
未能转换 javax.swing.JTable.JTable(TableModel)
未能转换 javax.swing.JTable.JTable(TableModel, TableColumnMode)
未能转换 javax.swing.JTable.JTable(TableModel, TableColumnModel, ListSelectionModel)
未能转换 javax.swing.JTable.JTable(Vector, Vector)
未能转换 javax.swing.JTable.moveColumn
未能转换 javax.swing.JTable paramString
未能转换 javax.swing.JTable.preferredViewportSize
未能转换 javax.swing.JTable.prepareEditor
未能转换 javax.swing.JTable.prepareRenderer
未能转换 javax.swing.JTable.processKeyBinding
未能转换 javax.swing.JTable.removeColumnSelectionInterval
未能转换 javax.swing.JTable.removeEditor
未能转换 javax.swing.JTable.removeRowSelectionInterval
未能转换 javax.swing.JTable.rowAtPoint
未能转换 javax.swing.JTable.rowHeight
未能转换 javax.swing.JTable.rowMargin
未能转换 javax.swing.JTable.rowSelectionAllowed
未能转换 javax.swing.JTable.selectAll
未能转换 javax.swing.JTable.selectionBackground
未能转换 javax.swing.JTable.selectionForeground
未能转换 javax.swing.JTable.selectionModel
未能转换 javax.swing.JTable.setAutoCreateColumnsFromModel
未能转换 javax.swing.JTable.setAutoResizeMode
未能转换 javax.swing.JTable.setCellEditor
未能转换 javax.swing.JTable.setCellSelectionEnabled
未能转换 javax.swing.JTable.setColumnModel
未能转换 javax.swing.JTable.setColumnSelectionAllowed
未能转换 javax.swing.JTable.setColumnSelectionInterval
未能转换 javax.swing.JTable.setComponentOrientation

未能转换 javax.swing.JTable.setDefaultEditor
未能转换 javax.swing.JTable.setDefaultRenderer
未能转换 javax.swing.JTable.setEditingColumn
未能转换 javax.swing.JTable.setEditingRow
未能转换 javax.swing.JTable.setIntercellSpacing
未能转换 javax.swing.JTable.setPreferredScrollableViewportSize
未能转换 javax.swing.JTable.setRowHeight
未能转换 javax.swing.JTable.setRowMargin
未能转换 javax.swing.JTable.setRowSelectionAllowed
未能转换 javax.swing.JTable.setRowSelectionInterval
未能转换 javax.swing.JTable.setSelectionBackground
未能转换 javax.swing.JTable.setSelectionForeground
未能转换 javax.swing.JTable.setSelectionMode
未能转换 javax.swing.JTable.setSelectionModel
未能转换 javax.swing.JTable.setShowGrid
未能转换 javax.swing.JTable.setShowHorizontalLines
未能转换 javax.swing.JTable.setShowVerticalLines
未能转换 javax.swing.JTable.setTableHeader
未能转换 javax.swing.JTable.setUI
未能转换 javax.swing.JTable.showHorizontalLines
未能转换 javax.swing.JTable.showVerticalLines
未能转换 javax.swing.JTable.sizeColumnsToFit
未能转换 javax.swing.JTable.tableChanged
未能转换 javax.swing.JTable.tableHeader
未能转换 javax.swing.JTable.unconfigureEnclosingScrollPane
未能转换 javax.swing.JTable.updateUI
未能转换 javax.swing.JTable.valueChanged
未能转换 javax.swing.JTextArea.createDefaultModel
未能转换 javax.swing.JTextArea.getColumns
未能转换 javax.swing.JTextArea getColumnWidth
未能转换 javax.swing.JTextArea.getLineEndOffset
未能转换 javax.swing.JTextArea.getLineOffset
未能转换 javax.swing.JTextArea.getLineStartOffset
未能转换 javax.swing.JTextArea.getPreferredScrollableViewportSize
未能转换 javax.swing.JTextArea.getRowHeight
未能转换 javax.swing.JTextArea.getRows
未能转换 javax.swing.JTextArea.getScrollableTracksViewportWidth
未能转换 javax.swing.JTextArea.getScrollableUnitIncrement
未能转换 javax.swing.JTextArea.getTabSize
未能转换 javax.swing.JTextArea.getUIClassID

未能转换 javax.swing.JTextArea.getWrapStyleWord
未能转换 javax.swing.JTextArea.isManagingFocus
未能转换 javax.swing.JTextArea.JTextArea(Document)
未能转换 javax.swing.JTextArea.JTextArea(Document, String, int, int)
未能转换 javax.swing.JTextArea.setColumns
未能转换 javax.swing.JTextArea.setRows
未能转换 javax.swing.JTextArea.setTabSize
未能转换 javax.swing.JTextField.configurePropertiesFromAction
未能转换 javax.swing.JTextField.createActionPropertyChangeListener
未能转换 javax.swing.JTextField.createDefaultModel
未能转换 javax.swing.JTextField.fireActionPerformed
未能转换 javax.swing.JTextField.getAction
未能转换 javax.swing.JTextField.getActions
未能转换 javax.swing.JTextField.getColumns
未能转换 javax.swing.JTextField.getColumnWidth
未能转换 javax.swing.JTextField.getHorizontalVisibility
未能转换 javax.swing.JTextField.getScrollOffset
未能转换 javax.swing.JTextField.getUIClassID
未能转换 javax.swing.JTextField.isValidateRoot
未能转换 javax.swing.JTextField.JTextField(Document, String, int)
未能转换 javax.swing.JTextField.JTextField(int)
未能转换 javax.swing.JTextField.JTextField(String, int)
未能转换 javax.swing.JTextField.notifyAction
未能转换 javax.swing.JTextField.postActionEvent
未能转换 javax.swing.JTextField.scrollRectToVisible
未能转换 javax.swing.JTextField.setActionCommand
未能转换 javax.swing.JTextField.setColumns
未能转换 javax.swing.JTextField.setScrollOffset
未能转换 javax.swing.JTextPane.addStyle
未能转换 javax.swing.JTextPane.createDefaultEditorKit
未能转换 javax.swing.JTextPane.getCharacterAttributes
未能转换 javax.swing.JTextPane.getInputAttributes
未能转换 javax.swing.JTextPane.getLogicalStyle
未能转换 javax.swing.JTextPane.getParagraphAttributes
未能转换 javax.swing.JTextPane.getStyle
未能转换 javax.swing.JTextPane.getStyledDocument
未能转换 javax.swing.JTextPane.getStyledEditorKit
未能转换 javax.swing.JTextPane.getUIClassID
未能转换 javax.swing.JTextPane.insertComponent
未能转换 javax.swing.JTextPane.insertIcon

未能转换 javax.swing.JTextPane.removeStyle
未能转换 javax.swing.JTextPane.setCharacterAttributes
未能转换 javax.swing.JTextPane.setDocument
未能转换 javax.swing.JTextPane.setEditorKit
未能转换 javax.swing.JTextPane.setLogicalStyle
未能转换 javax.swing.JTextPane.setParagraphAttributes
未能转换 javax.swing.JTextPane.setStyledDocument
未能转换 javax.swing.JToggleButton.getUIClassID
未能转换 javax.swing.JToggleButton.JToggleButton
未能转换 javax.swing.JToggleButton.ToggleButtonModel
未能转换 javax.swing.JToggleButton.updateUI
未能转换 javax.swing.JToolBar.addSeparator
未能转换 javax.swing.JToolBar.addSeparator(Dimension)
未能转换 javax.swing.JToolBar.createActionChangeListener
未能转换 javax.swing.JToolBar.getComponentAtIndex
未能转换 javax.swing.JToolBar.getComponentIndex
未能转换 javax.swing.JToolBar.getMargin
未能转换 javax.swing.JToolBar.getOrientation
未能转换 javax.swing.JToolBar.getUI
未能转换 javax.swing.JToolBar.getUIClassID
未能转换 javax.swing.JToolBar.isBorderPainted
未能转换 javax.swing.JToolBar.isFloatable
未能转换 javax.swing.JToolBar.paintBorder
未能转换 javax.swing.JToolBar.Separator
未能转换 javax.swing.JToolBar.setBorderPainted
未能转换 javax.swing.JToolBar.setFloatable
未能转换 javax.swing.JToolBar.setMargin
未能转换 javax.swing.JToolBar.setOrientation
未能转换 javax.swing.JToolBar.setUI
未能转换 javax.swing.JToolBar.updateUI
未能转换 javax.swing.JToolTip.getAccessibleContext
未能转换 javax.swing.JToolTip.getComponent
未能转换 javax.swing.JToolTip.getText
未能转换 javax.swing.JToolTip.getUI
未能转换 javax.swing.JToolTip.getUIClassID
未能转换 javax.swing.JToolTip paramString
未能转换 javax.swing.JToolTip.setComponent
未能转换 javax.swing.JToolTip.setText
未能转换 javax.swing.JToolTip.updateUI
未能转换 javax.swing.JTree

未能转换 javax.swing.JTree.<PropertyName>
未能转换 javax.swing.JTree.addSelectionInterval
未能转换 javax.swing.JTree.addSelectionPath
未能转换 javax.swing.JTree.addSelectionPaths
未能转换 javax.swing.JTree.addSelectionRow
未能转换 javax.swing.JTree.addSelectionRows
未能转换 javax.swing.JTree.cancelEditing
未能转换 javax.swing.JTree.cellEditor
未能转换 javax.swing.JTree.cellRenderer
未能转换 javax.swing.JTree.clearToggledPaths
未能转换 javax.swing.JTree.collapsePath
未能转换 javax.swing.JTree.collapseRow
未能转换 javax.swing.JTree.convertValueToText
未能转换 javax.swing.JTree.createTreeModel
未能转换 javax.swing.JTree.createTreeModelListener
未能转换 javax.swing.JTree.DynamicUtilTreeNode
未能转换 javax.swing.JTree.expandPath
未能转换 javax.swing.JTree.expandRow
未能转换 javax.swing.JTree.fireTreeCollapsed
未能转换 javax.swing.JTree.fireTreeExpanded
未能转换 javax.swing.JTree.fireTreeWillCollapse
未能转换 javax.swing.JTree.fireTreeWillExpand
未能转换 javax.swing.JTree.fireValueChanged
未能转换 javax.swing.JTree.getAnchorSelectionPath
未能转换 javax.swing.JTree.getCellEditor
未能转换 javax.swing.JTree.getCellRenderer
未能转换 javax.swing.JTree.getClosestPathForLocation
未能转换 javax.swing.JTree.getClosestRowForLocation
未能转换 javax.swing.JTree.getDefaultTreeModel
未能转换 javax.swing.JTree.getDescendantToggledPaths
未能转换 javax.swing.JTree.getEditingPath
未能转换 javax.swing.JTree.getExpandedDescendants
未能转换 javax.swing.JTree.getExpandsSelectedPaths
未能转换 javax.swing.JTree.getInvokesStopCellEditing
未能转换 javax.swing.JTree.getLeadSelectionPath
未能转换 javax.swing.JTree.getLeadSelectionRow
未能转换 javax.swing.JTree.getMaxSelectionRow
未能转换 javax.swing.JTree.getMinSelectionRow
未能转换 javax.swing.JTree.getModel
未能转换 javax.swing.JTree.getPathBetweenRows

未能转换 javax.swing.JTree.getPathBounds
未能转换 javax.swing.JTree.getPathForLocation
未能转换 javax.swing.JTree.getPathForRow
未能转换 javax.swing.JTree.getPreferredScrollableViewportSize
未能转换 javax.swing.JTree.getRowBounds
未能转换 javax.swing.JTree.getRowCount
未能转换 javax.swing.JTree.getRowForLocation
未能转换 javax.swing.JTree.getRowForPath
未能转换 javax.swing.JTree.getRowHeight
未能转换 javax.swing.JTree.getScrollableBlockIncrement
未能转换 javax.swing.JTree.getScrollableTracksViewportHeight
未能转换 javax.swing.JTree.getScrollableTracksViewportWidth
未能转换 javax.swing.JTree.getScrollableUnitIncrement
未能转换 javax.swing.JTree.getScrollsOnExpand
未能转换 javax.swing.JTree.getSelectionCount
未能转换 javax.swing.JTree.getSelectionModel
未能转换 javax.swing.JTree.getSelectionPath
未能转换 javax.swing.JTree.getSelectionPaths
未能转换 javax.swing.JTree.getSelectionRows
未能转换 javax.swing.JTree.getToggleClickCount
未能转换 javax.swing.JTree.getToolTipText
未能转换 javax.swing.JTree.getUI
未能转换 javax.swing.JTree.getUIClassID
未能转换 javax.swing.JTree.hasBeenExpanded
未能转换 javax.swing.JTree.invokesStopCellEditing
未能转换 javax.swing.JTree.isCollapsed(int)
未能转换 javax.swing.JTree.isCollapsed(TreePath)
未能转换 javax.swing.JTree.isEditing
未能转换 javax.swing.JTree.isExpanded(int)
未能转换 javax.swing.JTree.isExpanded(TreePath)
未能转换 javax.swing.JTree.isFixedRowHeight
未能转换 javax.swing.JTree.isLargeModel
未能转换 javax.swing.JTree.isPathEditable
未能转换 javax.swing.JTree.isPathSelected
未能转换 javax.swing.JTree.isRootVisible
未能转换 javax.swing.JTree.isRowSelected
未能转换 javax.swing.JTree.isVisible
未能转换 javax.swing.JTree.JTree(Hashtable)
未能转换 javax.swing.JTree.JTree(TreeNode, boolean)
未能转换 javax.swing.JTree.largeModel

未能转换 javax.swing.JTree.makeVisible
未能转换 javax.swing.JTree.removeDescendantSelectedPaths
未能转换 javax.swing.JTree.removeDescendantToggledPaths
未能转换 javax.swing.JTree.removeSelectionInterval
未能转换 javax.swing.JTree.removeSelectionPath
未能转换 javax.swing.JTree.removeSelectionPaths
未能转换 javax.swing.JTree.removeSelectionRow
未能转换 javax.swing.JTree.removeSelectionRows
未能转换 javax.swing.JTree.rootVisible
未能转换 javax.swing.JTree.scrollPathToVisible
未能转换 javax.swing.JTree.scrollRowToVisible
未能转换 javax.swing.JTree.scrollsOnExpand
未能转换 javax.swing.JTree.selectionModel
未能转换 javax.swing.JTree.selectionRedirector
未能转换 javax.swing.JTree.setAnchorSelectionPath
未能转换 javax.swing.JTree.setCellEditor
未能转换 javax.swing.JTree.setCellRenderer
未能转换 javax.swing.JTree.setExpandedState
未能转换 javax.swing.JTree.setExpandsSelectedPaths
未能转换 javax.swing.JTree.setInvokesStopCellEditing
未能转换 javax.swing.JTree.setLargeModel
未能转换 javax.swing.JTree.setLeadSelectionPath
未能转换 javax.swing.JTree.setRootVisible
未能转换 javax.swing.JTree.setRowHeight
未能转换 javax.swing.JTree.setScrollsOnExpand
未能转换 javax.swing.JTree.setSelectionInterval
未能转换 javax.swing.JTree.setSelectionModel
未能转换 javax.swing.JTree.setSelectionPath
未能转换 javax.swing.JTree.setSelectionPaths
未能转换 javax.swing.JTree.setSelectionRow
未能转换 javax.swing.JTree.setSelectionRows
未能转换 javax.swing.JTree.setToggleClickCount
未能转换 javax.swing.JTree.setUI
未能转换 javax.swing.JTree.setVisibleRowCount
未能转换 javax.swing.JTree.startEditingAtPath
未能转换 javax.swing.JTree.stopEditing
未能转换 javax.swing.JTree.toggleClickCount
未能转换 javax.swing.JTree.treeDidChange
未能转换 javax.swing.JTree.treeModel
未能转换 javax.swing.JTree.treeModelListener

未能转换 javax.swing.JTree.updateUI
未能转换 javax.swing.JTree.visibleRowCount
未能转换 javax.swing.JViewport
未能转换 javax.swing.JWindow.getGlassPane
未能转换 javax.swing.JWindow.getLayeredPane
未能转换 javax.swing.JWindow.isRootPaneCheckingEnabled
未能转换 javax.swing.JWindow.JWindow(GraphicsConfiguration)
未能转换 javax.swing.JWindow.JWindow(GraphicsConfiguration, Window)
未能转换 javax.swing.JWindow.rootPane
未能转换 javax.swing.JWindow.rootPaneCheckingEnabled
未能转换 javax.swing.JWindow.setContentPane
未能转换 javax.swing.JWindow.setGlassPane
未能转换 javax.swing.JWindow.setLayeredPane
未能转换 javax.swing.JWindow.setRootPane
未能转换 javax.swing.JWindow.setRootPaneCheckingEnabled
未能转换 javax.swing.JWindow.windowInit
未能转换 javax.swing.KeyStroke.getKeyStroke
未能转换 javax.swing.KeyStroke.getKeyStrokeForEvent
未能转换 javax.swing.KeyStroke.isOnKeyRelease
未能转换 javax.swing.KeyStroke.KeyStroke
未能转换 javax.swing.ListCellRenderer
未能转换 javax.swingListModel
未能转换 javax.swing.ListModel.addDataListListener
未能转换 javax.swing.ListModel.removeDataListListener
未能转换 javax.swing.ListSelectionModel
未能转换 javax.swing.ListSelectionModel.addDataSelectionListener
未能转换 javax.swing.ListSelectionModel.getValueIsAdjusting
未能转换 javax.swing.ListSelectionModel.insertIndexInterval
未能转换 javax.swing.ListSelectionModel.removeDataSelectionListener
未能转换 javax.swing.ListSelectionModel.setValueIsAdjusting
未能转换 javax.swing.LookAndFeel
未能转换 javax.swing.MenuElement.menuSelectionChanged
未能转换 javax.swing.MenuSelectionManager
未能转换 javax.swing.OverlayLayout
未能转换 javax.swing.ProgressMonitor
未能转换 javax.swing.ProgressMonitorInputStream
未能转换 javax.swing.Renderer
未能转换 javax.swing.RepaintManager
未能转换 javax.swing.RootPaneContainer.getGlassPane
未能转换 javax.swing.RootPaneContainer.getLayeredPane

未能转换 javax.swing.RootPaneContainer.setContentPane
未能转换 javax.swing.RootPaneContainer.setGlassPane
未能转换 javax.swing.RootPaneContainer.setLayeredPane
未能转换 javax.swing.Scrollable
未能转换 javax.swing.ScrollPaneConstants
未能转换 javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
未能转换 javax.swing.ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER
未能转换 javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
未能转换 javax.swing.ScrollPaneConstants.VERTICAL_SCROLLBAR_NEVER
未能转换 javax.swing.ScrollPaneLayout
未能转换 javax.swing.ScrollPaneLayout.UIResource
未能转换 javax.swing.SingleSelectionModel
未能转换 javax.swing.SingleSelectionModel.addChangeListener
未能转换 javax.swing.SingleSelectionModel.removeChangeListener
未能转换 javax.swing.SizeRequirements
未能转换 javax.swing.SizeSequence
未能转换 javax.swing.SizeSequence.getIndex
未能转换 javax.swing.SizeSequence getPosition
未能转换 javax.swing.SizeSequence getSize
未能转换 javax.swing.SizeSequence getSizes
未能转换 javax.swing.SizeSequence.insertEntries
未能转换 javax.swing.SizeSequence.removeEntries
未能转换 javax.swing.SizeSequence setSize
未能转换 javax.swing.SizeSequence setSizes
未能转换 javax.swing.SizeSequence.SizeSequence
未能转换 javax.swing.SwingConstants
未能转换 javax.swing.SwingUtilities
未能转换 javax.swing.SwingUtilities.getAccessibleAt
未能转换 javax.swing.SwingUtilities.getAncestorNamed
未能转换 javax.swing.SwingUtilities.getDeepestComponentAt
未能转换 javax.swing.SwingUtilities.getLocalBounds
未能转换 javax.swing.SwingUtilities.getRoot
未能转换 javax.swing.SwingUtilities.getRootPane
未能转换 javax.swing.SwingUtilities.getWindowAncestor
未能转换 javax.swing.Timer.fireActionPerformed
未能转换 javax.swing.Timer.getInitialDelay
未能转换 javax.swing.Timer.getListeners
未能转换 javax.swing.Timer.getLogTimers
未能转换 javax.swing.Timer.isCoalesce
未能转换 javax.swing.Timer.restart

未能转换 javax.swing.Timer.setCoalesce
未能转换 javax.swing.Timer.setInitialDelay
未能转换 javax.swing.Timer.setLogTimers
未能转换 javax.swing.ToolTipManager.heavyWeightPopupEnabled
未能转换 javax.swing.ToolTipManager.isLightWeightPopupEnabled
未能转换 javax.swing.ToolTipManager.lightWeightPopupEnabled
未能转换 javax.swing.ToolTipManager.mouseDragged
未能转换 javax.swing.ToolTipManager.mouseEntered
未能转换 javax.swing.ToolTipManager.mouseExited
未能转换 javax.swing.ToolTipManager.mouseMoved
未能转换 javax.swing.ToolTipManager.mousePressed
未能转换 javax.swing.ToolTipManager.registerComponent
未能转换 javax.swing.ToolTipManager.setLightWeightPopupEnabled
未能转换 javax.swing.ToolTipManager.ToolTipManager
未能转换 javax.swing.ToolTipManager.unregisterComponent
未能转换 javax.swing.UIDefaults
未能转换 javax.swing.UIDefaults.ActiveValue
未能转换 javax.swing.UIDefaults.LazyInputMap
未能转换 javax.swing.UIDefaults.LazyValue
未能转换 javax.swing.UIDefaults.ProxyLazyValue
未能转换 javax.swing.UIManager
未能转换 javax.swing.UIManager.LookAndFeelInfo
未能转换 javax.swing.UnsupportedLookAndFeelException
未能转换 javax.swing.ViewportLayout
未能转换 javax.swing.WindowConstants

Javax.swing.beaninfo 错误信息

未能转换 javax.swing.beaninfo.SwingBeanInfo

Javax.swing.border 错误信息

未能转换 javax.swing.border.AbstractBorder
未能转换 javax.swing.border.BevelBorder
未能转换 javax.swing.border.BevelBorder.<Type>
未能转换 javax.swing.border.BevelBorder.BevelBorder
未能转换 javax.swing.border.BevelBorder.bevelType
未能转换 javax.swing.border.BevelBorder.getBevelType
未能转换 javax.swing.border.BevelBorder.getBorderInsets
未能转换 javax.swing.border.BevelBorder.getHighlightInnerColor
未能转换 javax.swing.border.BevelBorder.getHighlightOuterColor
未能转换 javax.swing.border.BevelBorder.getShadowInnerColor
未能转换 javax.swing.border.BevelBorder.getShadowOuterColor
未能转换 javax.swing.border.BevelBorder.highlightInner
未能转换 javax.swing.border.BevelBorder.highlightOuter
未能转换 javax.swing.border.BevelBorder.isBorderOpaque
未能转换 javax.swing.border.BevelBorder.paintBorder
未能转换 javax.swing.border.BevelBorder.paintLoweredBevel
未能转换 javax.swing.border.BevelBorder.paintRaisedBevel
未能转换 javax.swing.border.BevelBorder.shadowInner
未能转换 javax.swing.border.BevelBorder.shadowOuter
未能转换 javax.swing.border.Border
未能转换 javax.swing.border.Border.getBorderInsets
未能转换 javax.swing.border.Border.isBorderOpaque
未能转换 javax.swing.border.Border.paintBorder
未能转换 javax.swing.border.CompoundBorder
未能转换 javax.swing.border.EmptyBorder
未能转换 javax.swing.border.EmptyBorder.bottom
未能转换 javax.swing.border.EmptyBorder.EmptyBorder
未能转换 javax.swing.border.EmptyBorder.getBorderInsets
未能转换 javax.swing.border.EmptyBorder.isBorderOpaque
未能转换 javax.swing.border.EmptyBorder.left
未能转换 javax.swing.border.EmptyBorder.paintBorder
未能转换 javax.swing.border.EmptyBorder.right
未能转换 javax.swing.border.EmptyBorder.top
未能转换 javax.swing.border.EtchedBorder
未能转换 javax.swing.border.EtchedBorder.<Type>
未能转换 javax.swing.border.EtchedBorder.EtchedBorder
未能转换 javax.swing.border.EtchedBorder.etchType

未能转换 javax.swing.border.EtchedBorder.getBorderInsets
未能转换 javax.swing.border.EtchedBorder.getEtchType
未能转换 javax.swing.border.EtchedBorder.getHighlightColor
未能转换 javax.swing.border.EtchedBorder.getShadowColor
未能转换 javax.swing.border.EtchedBorder.highlight
未能转换 javax.swing.border.EtchedBorder.isBorderOpaque
未能转换 javax.swing.border.EtchedBorder.paintBorder
未能转换 javax.swing.border.EtchedBorder.shadow
未能转换 javax.swing.border.LineBorder
未能转换 javax.swing.border.LineBorder.createBlackLineBorder
未能转换 javax.swing.border.LineBorder.createGrayLineBorder
未能转换 javax.swing.border.LineBorder.getBorderInsets
未能转换 javax.swing.border.LineBorder.getLineColor
未能转换 javax.swing.border.LineBorder.getRoundedCorners
未能转换 javax.swing.border.LineBorder.getThickness
未能转换 javax.swing.border.LineBorder.isBorderOpaque
未能转换 javax.swing.border.LineBorder.LineBorder
未能转换 javax.swing.border.LineBorder.lineColor
未能转换 javax.swing.border.LineBorder.paintBorder
未能转换 javax.swing.border.LineBorder.roundedCorners
未能转换 javax.swing.border.LineBorder.thickness
未能转换 javax.swing.border.MatteBorder
未能转换 javax.swing.border.SoftBevelBorder
未能转换 javax.swing.border.SoftBevelBorder.getBorderInsets
未能转换 javax.swing.border.SoftBevelBorder.isBorderOpaque
未能转换 javax.swing.border.SoftBevelBorder.paintBorder
未能转换 javax.swing.border.SoftBevelBorder.SoftBevelBorder
未能转换 javax.swing.border.TitledBorder

Javax.swing.colorchooser 错误信息

未能转换 javax.swing.colorchooser.AbstractColorChooserPanel
未能转换 javax.swing.colorchooser.ColorChooserComponentFactory
未能转换 javax.swing.colorchooser.ColorSelectionModel
未能转换 javax.swing.colorchooser.DefaultColorSelectionModel

Javax.swing.event 错误信息

未能转换 javax.swing.event.AncestorEvent
未能转换 javax.swing.event.CaretEvent
未能转换 javax.swing.event.DocumentEvent
未能转换 javax.swing.event.DocumentEvent.ElementChange
未能转换 javax.swing.event.DocumentEvent.EventType
未能转换 javax.swing.event.EventListenerList.getListenerCount
未能转换 javax.swing.event.EventListenerList.getListeners
未能转换 javax.swing.event.HyperlinkEvent.EventType
未能转换 javax.swing.event.HyperlinkEvent.getDescription
未能转换 javax.swing.event.HyperlinkEvent.getEventType
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_ACTIVATED
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_CLOSED
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_CLOSING
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_DEACTIVATED
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_DEICONIFIED
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_FIRST
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_ICONIFIED
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_LAST
未能转换 javax.swing.event.InternalFrameEvent.INTERNAL_FRAME_OPENED
未能转换 javax.swing.event.ListDataEvent
未能转换 javax.swing.event.ListSelectionEvent.getFirstIndex
未能转换 javax.swing.event.ListSelectionEvent.getLastIndex
未能转换 javax.swing.event.ListSelectionEvent.getValueIsAdjusting
未能转换 javax.swing.event.MenuDragMouseEvent
未能转换 javax.swing.event.MenuKeyEvent
未能转换 javax.swing.event.SwingPropertyChangeEvent
未能转换 javax.swing.event.TableColumnModelEvent.fromIndex
未能转换 javax.swing.event.TableColumnModelEvent.getFromIndex
未能转换 javax.swing.event.TableColumnModelEvent.getToIndex
未能转换 javax.swing.event.TableColumnModelEvent.TableColumnModelEvent
未能转换 javax.swing.event.TableColumnModelEvent.toIndex
未能转换 javax.swing.event.TableModelEvent.ALL_COLUMNS
未能转换 javax.swing.event.TableModelEvent.column
未能转换 javax.swing.event.TableModelEvent.firstRow
未能转换 javax.swing.event.TableModelEvent.getColumn
未能转换 javax.swing.event.TableModelEvent.getFirstRow
未能转换 javax.swing.event.TableModelEvent getLastRow

未能转换 javax.swing.event.TableModelEvent.HEADER_ROW
未能转换 javax.swing.event.TableModelEvent.lastRow
未能转换 javax.swing.event.TableModelEvent.TableModelEvent
未能转换 javax.swing.event.TreeExpansionEvent.getPath
未能转换 javax.swing.event.TreeExpansionEvent.path
未能转换 javax.swing.event.TreeExpansionEvent.TreeExpansionEvent
未能转换 javax.swing.event.TreeModelEvent
未能转换 javax.swing.event.TreeSelectionEvent.areNew
未能转换 javax.swing.event.TreeSelectionEvent.cloneWithSource
未能转换 javax.swing.event.TreeSelectionEvent.getNewLeadSelectionPath
未能转换 javax.swing.event.TreeSelectionEvent.getOldLeadSelectionPath
未能转换 javax.swing.event.TreeSelectionEvent.getPath
未能转换 javax.swing.event.TreeSelectionEvent.getPaths
未能转换 javax.swing.event.TreeSelectionEvent.isAddedPath
未能转换 javax.swing.event.TreeSelectionEvent.newLeadSelectionPath
未能转换 javax.swing.event.TreeSelectionEvent.oldLeadSelectionPath
未能转换 javax.swing.event.TreeSelectionEvent.paths
未能转换 javax.swing.event.TreeSelectionEvent.TreeSelectionEvent
未能转换 javax.swing.event.UndoableEditEvent

Javax.swing.filechooser 错误信息

未能转换 javax.swing.filechooser.FileFilter

未能转换 javax.swing.filechooser.FileSystemView.createNewFolder

未能转换 javax.swing.filechooser.FileSystemView.FileSystemView

未能转换 javax.swing.filechooser.FileSystemView.getFileSystemView

未能转换 javax.swing.filechooser.FileSystemView.getHomeDirectory

未能转换 javax.swing.filechooser.FileSystemView.getRoots

未能转换 javax.swing.filechooser.FileSystemView.isRoot

未能转换 javax.swing.filechooser.FileView

Javax.swing.table 错误信息

未能转换 javax.swing.table.AbstractTableModel
未能转换 javax.swing.table.AbstractTableModel.addTableModelListener
未能转换 javax.swing.table.AbstractTableModel.fireTableCellUpdated
未能转换 javax.swing.table.AbstractTableModel.fireTableChanged
未能转换 javax.swing.table.AbstractTableModel.fireTableDataChanged
未能转换 javax.swing.table.AbstractTableModel.fireTableRowsDeleted
未能转换 javax.swing.table.AbstractTableModel.fireTableRowsInserted
未能转换 javax.swing.table.AbstractTableModel.fireTableRowsUpdated
未能转换 javax.swing.table.AbstractTableModel.fireTableStructureChanged
未能转换 javax.swing.table.AbstractTableModel.getListeners
未能转换 javax.swing.table.AbstractTableModel.listenerList
未能转换 javax.swing.table.AbstractTableModel.removeTableModelListener
未能转换 javax.swing.table.DefaultTableCellRenderer
未能转换 javax.swing.table.DefaultTableCellRenderer.UIResource
未能转换 javax.swing.table.DefaultTableModel
未能转换 javax.swing.table.DefaultTableModel.addColumnModelListener
未能转换 javax.swing.table.DefaultTableModel.changeEvent
未能转换 javax.swing.table.DefaultTableModel.columnMargin
未能转换 javax.swing.table.DefaultTableModel.columnSelectionAllowed
未能转换 javax.swing.table.DefaultTableModel.createSelectionModel
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel
未能转换 javax.swing.table.DefaultTableModel.fireColumnAdded
未能转换 javax.swing.table.DefaultTableModel.fireColumnMarginChanged
未能转换 javax.swing.table.DefaultTableModel.fireColumnMoved
未能转换 javax.swing.table.DefaultTableModel.fireColumnRemoved
未能转换 javax.swing.table.DefaultTableModel.fireColumnSelectionChanged
未能转换 javax.swing.table.DefaultTableModel.getColumnIndexAtX
未能转换 javax.swing.table.DefaultTableModel.getColumnMargin
未能转换 javax.swing.table.DefaultTableModel.getColumnSelectionAllowed
未能转换 javax.swing.table.DefaultTableModel.getListeners
未能转换 javax.swing.table.DefaultTableModel.getSelectedColumnCount
未能转换 javax.swing.table.DefaultTableModel.getSelectedColumns
未能转换 javax.swing.table.DefaultTableModel.getSelectionModel
未能转换 javax.swing.table.DefaultTableModel.getTotalColumnWidth
未能转换 javax.swing.table.DefaultTableModel.listenerList
未能转换 javax.swing.table.DefaultTableModel.moveColumn
未能转换 javax.swing.table.DefaultTableModel.propertyChange

未能转换 javax.swing.table.DefaultTableModel.recalcWidthCache
未能转换 javax.swing.table.DefaultTableModel.removeColumnModelListener
未能转换 javax.swing.table.DefaultTableModel.selectionModel
未能转换 javax.swing.table.DefaultTableModel.setColumnMargin
未能转换 javax.swing.table.DefaultTableModel.setColumnSelectionAllowed
未能转换 javax.swing.table.DefaultTableModel.setSelectionModel
未能转换 javax.swing.table.DefaultTableModel.tableColumns
未能转换 javax.swing.table.DefaultTableModel.totalColumnWidth
未能转换 javax.swing.table.DefaultTableModel.valueChanged
未能转换 javax.swing.table.DefaultTableModel
未能转换 javax.swing.table.DefaultTableModel.addColumn(Object, Object[])
未能转换 javax.swing.table.DefaultTableModel.addColumn(Object, Vector)
未能转换 javax.swing.table.DefaultTableModel.columnIdentifiers
未能转换 javax.swing.table.DefaultTableModel.dataVector
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel(int, int)
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel(Object[], int)
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel(Object[][][], Object[][])
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel(Vector, int)
未能转换 javax.swing.table.DefaultTableModel.DefaultTableModel(Vector, Vector)
未能转换 javax.swing.table.DefaultTableModel.getDataVector
未能转换 javax.swing.table.DefaultTableModel.insertRow
未能转换 javax.swing.table.DefaultTableModel.isCellEditable
未能转换 javax.swing.table.DefaultTableModel.moveRow
未能转换 javax.swing.table.DefaultTableModel.newDataAvailable
未能转换 javax.swing.table.DefaultTableModel.newRowsAdded
未能转换 javax.swing.table.DefaultTableModel.rowsRemoved
未能转换 javax.swing.table.DefaultTableModel.setRowCount
未能转换 javax.swing.table.DefaultTableModel.setColumnIdentifiers
未能转换 javax.swing.table.DefaultTableModel.setDataVector
未能转换 javax.swing.table.DefaultTableModel.setNumRows
未能转换 javax.swing.table.DefaultTableModel.setRowCount
未能转换 javax.swing.table.JTableHeader
未能转换 javax.swing.table.TableCellEditor
未能转换 javax.swing.table.TableCellRenderer
未能转换 javax.swing.table.TableColumn
未能转换 javax.swing.table.TableColumn.<PropertyName>
未能转换 javax.swing.table.TableColumn.addPropertyChangeListener
未能转换 javax.swing.table.TableColumn.disableResizedPosting
未能转换 javax.swing.table.TableColumn.enableResizedPosting
未能转换 javax.swing.table.TableColumn.getHeaderValue

未能转换 javax.swing.table.TableColumn.getIdentifier
未能转换 javax.swing.table.TableColumn.getMaxWidth
未能转换 javax.swing.table.TableColumn.getMinWidth
未能转换 javax.swing.table.TableColumn.getModelIndex
未能转换 javax.swing.table.TableColumn.getPreferredWidth
未能转换 javax.swing.table.TableColumn.getResizable
未能转换 javax.swing.table.TableColumn.getWidth
未能转换 javax.swing.table.TableColumn.headerValue
未能转换 javax.swing.table.TableColumn.identifier
未能转换 javax.swing.table.TableColumn.isResizable
未能转换 javax.swing.table.TableColumn.maxWidth
未能转换 javax.swing.table.TableColumn.minWidth
未能转换 javax.swing.table.TableColumn.modelIndex
未能转换 javax.swing.table.TableColumn.removePropertyChangeListener
未能转换 javax.swing.table.TableColumn.resizedPostingDisableCount
未能转换 javax.swing.table.TableColumn.setHeaderValue
未能转换 javax.swing.table.TableColumn.setIdentifier
未能转换 javax.swing.table.TableColumn.setMaxWidth
未能转换 javax.swing.table.TableColumn.setMinWidth
未能转换 javax.swing.table.TableColumn.setModelIndex
未能转换 javax.swing.table.TableColumn.setPreferredWidth
未能转换 javax.swing.table.TableColumn.setResizable
未能转换 javax.swing.table.TableColumn.setWidth
未能转换 javax.swing.table.TableColumn.sizeWidthToFit
未能转换 javax.swing.table.TableColumn.TableColumn
未能转换 javax.swing.table.TableColumn.width
未能转换 javax.swing.table.TableColumnModel
未能转换 javax.swing.table.TableColumnModel.getColumnIndexAtX
未能转换 javax.swing.table.TableColumnModel.getColumnMargin
未能转换 javax.swing.table.TableColumnModel.addColumnModelListener
未能转换 javax.swing.table.TableColumnModel.getColumnSelectionAllowed
未能转换 javax.swing.table.TableColumnModel.getSelectedColumnCount
未能转换 javax.swing.table.TableColumnModel.getSelectedColumns
未能转换 javax.swing.table.TableColumnModel.getSelectionModel
未能转换 javax.swing.table.TableColumnModel.getTotalColumnWidth
未能转换 javax.swing.table.TableColumnModel.moveColumn
未能转换 javax.swing.table.TableColumnModel.removeColumnModelListener
未能转换 javax.swing.table.TableColumnModel.setColumnMargin
未能转换 javax.swing.table.TableColumnModel.setColumnSelectionAllowed
未能转换 javax.swing.table.TableColumnModel.setSelectionModel

未能转换 javax.swing.table.TableModel

未能转换 javax.swing.table.TableModel.addTableModelListener

未能转换 javax.swing.table.TableModel.removeTableModelListener

Javax.swing.text 错误信息

未能转换 javax.swing.text.AbstractDocument
未能转换 javax.swing.text.AbstractDocument.AbstractElement
未能转换 javax.swing.text.AbstractDocument.AttributeContext
未能转换 javax.swing.text.AbstractDocument.BranchElement
未能转换 javax.swing.text.AbstractDocument.Content
未能转换 javax.swing.text.AbstractDocument.DefaultDocumentEvent
未能转换 javax.swing.text.AbstractDocument.ElementEdit
未能转换 javax.swing.text.AbstractDocument.LeafElement
未能转换 javax.swing.text.AbstractWriter.AbstractWriter
未能转换 javax.swing.text.AbstractWriter.decrIndent
未能转换 javax.swing.text.AbstractWriter.getCanWrapLines
未能转换 javax.swing.text.AbstractWriter.getCurrentLineLength
未能转换 javax.swing.text.AbstractWriter.getDocument
未能转换 javax.swing.text.AbstractWriter.getElementIterator
未能转换 javax.swing.text.AbstractWriter.getEndOffset
未能转换 javax.swing.text.AbstractWriter.getIndentLevel
未能转换 javax.swing.text.AbstractWriter.getIndentSpace
未能转换 javax.swing.text.AbstractWriter.getLineLength
未能转换 javax.swing.text.AbstractWriter.getStartOffset
未能转换 javax.swing.text.AbstractWriter.getText
未能转换 javax.swing.text.AbstractWriter.getWriter
未能转换 javax.swing.text.AbstractWriter.incrIndent
未能转换 javax.swing.text.AbstractWriter.indent
未能转换 javax.swing.text.AbstractWriter.inRange
未能转换 javax.swing.text.AbstractWriter.isEmpty
未能转换 javax.swing.text.AbstractWriter.NEWLINE
未能转换 javax.swing.text.AbstractWriter.setCanWrapLines
未能转换 javax.swing.text.AbstractWriter.setCurrentLineLength
未能转换 javax.swing.text.AbstractWriter.setIndentSpace
未能转换 javax.swing.text.AbstractWriter.setLineLength
未能转换 javax.swing.text.AbstractWriter.text
未能转换 javax.swing.text.AbstractWriter.writeAttributes
未能转换 javax.swing.text.AsyncBoxView
未能转换 javax.swing.text.AsyncBoxView.ChildLocator
未能转换 javax.swing.text.AsyncBoxView.ChildState
未能转换 javax.swing.text.AttributeSet.CharacterAttribute
未能转换 javax.swing.text.AttributeSet.ColorAttribute

未能转换 javax.swing.text.AttributeSet.copyAttributes
未能转换 javax.swing.text.AttributeSet.FontAttribute
未能转换 javax.swing.text.AttributeSet.getResolveParent
未能转换 javax.swing.text.AttributeSet.isDefined
未能转换 javax.swing.text.AttributeSet isEqual
未能转换 javax.swing.text.AttributeSet.NameAttribute
未能转换 javax.swing.text.AttributeSet.ParagraphAttribute
未能转换 javax.swing.text.AttributeSet.ResolveAttribute
未能转换 javax.swing.text.BadLocationException
未能转换 javax.swing.text.BadLocationException.offsetRequested
未能转换 javax.swing.text.BoxView
未能转换 javax.swing.text.Caret
未能转换 javax.swing.text.ChangedCharSetException
未能转换 javax.swing.text.ChangedCharSetException.keyEqualsCharSet
未能转换 javax.swing.text.ComponentView
未能转换 javax.swing.text.CompositeView
未能转换 javax.swing.text.DefaultCaret
未能转换 javax.swing.text.DefaultEditorKit
未能转换 javax.swing.text.DefaultEditorKit.BeepAction
未能转换 javax.swing.text.DefaultEditorKit.CopyAction
未能转换 javax.swing.text.DefaultEditorKit.CutAction
未能转换 javax.swing.text.DefaultEditorKit.DefaultKeyTypedAction
未能转换 javax.swing.text.DefaultEditorKit.InsertBreakAction
未能转换 javax.swing.text.DefaultEditorKit.InsertContentAction
未能转换 javax.swing.text.DefaultEditorKit.InsertTabAction
未能转换 javax.swing.text.DefaultEditorKit.PasteAction
未能转换 javax.swing.text.DefaultHighlighter
未能转换 javax.swing.text.DefaultHighlighter.DefaultHighlightPainter
未能转换 javax.swing.text.DefaultStyledDocument
未能转换 javax.swing.text.DefaultStyledDocument.AttributeUndoableEdit
未能转换 javax.swing.text.DefaultStyledDocument.ElementBuffer
未能转换 javax.swing.text.DefaultStyledDocument.ElementSpec
未能转换 javax.swing.text.DefaultTextUI
未能转换 javax.swing.text.Document
未能转换 javax.swing.text.EditorKit
未能转换 javax.swing.text.Element
未能转换 javax.swing.text.ElementIterator
未能转换 javax.swing.text.FieldView
未能转换 javax.swing.text.FlowView
未能转换 javax.swing.text.FlowView.FlowStrategy

未能转换 javax.swing.text.GapContent
未能转换 javax.swing.text.GlyphView
未能转换 javax.swing.text.GlyphView.GlyphPainter
未能转换 javax.swing.text.Highlighter
未能转换 javax.swing.text.Highlighter.Highlight
未能转换 javax.swing.text.Highlighter.HighlightPainter
未能转换 javax.swing.text.html.BlockView
未能转换 javax.swing.text.html.CSS
未能转换 javax.swing.text.html.CSS.Attribute.<AttributeName>
未能转换 javax.swing.text.html.CSS.Attribute.getDefaultValue
未能转换 javax.swing.text.html.CSS.Attribute.isInherited
未能转换 javax.swing.text.html.FormView
未能转换 javax.swing.text.html.HTML.Attribute.<AttributeName>
未能转换 javax.swing.text.html.HTML.Tag.<ElementName>
未能转换 javax.swing.text.html.HTML.Tag.breaksFlow
未能转换 javax.swing.text.html.HTML.Tag.isBlock
未能转换 javax.swing.text.html.HTML.Tag.isPreformatted
未能转换 javax.swing.text.html.HTML.Tag.Tag
未能转换 javax.swing.text.html.HTML.UnknownTag
未能转换 javax.swing.text.html.HTMLDocument.AdditionalComments
未能转换 javax.swing.text.html.HTMLDocument.BlockElement.BlockElement
未能转换 javax.swing.text.html.HTMLDocument.BlockElement.getResolveParent
未能转换 javax.swing.text.html.HTMLDocument.create
未能转换 javax.swing.text.html.HTMLDocument.createBranchElement
未能转换 javax.swing.text.html.HTMLDocument.createDefaultRoot
未能转换 javax.swing.text.html.HTMLDocument.createLeafElement
未能转换 javax.swing.text.html.HTMLDocument.fireChangedUpdate
未能转换 javax.swing.text.html.HTMLDocument.fireUndoableEditUpdate
未能转换 javax.swing.text.html.HTMLDocument.getElement
未能转换 javax.swing.text.html.HTMLDocument.getParser
未能转换 javax.swing.text.html.HTMLDocument.getPreservesUnknownTags
未能转换 javax.swing.text.html.HTMLDocument.getReader
未能转换 javax.swing.text.html.HTMLDocument.getTokenThreshold
未能转换 javax.swing.text.html.HTMLDocument.HTMLDocument
未能转换 javax.swing.text.html.HTMLDocument.HTMLDocument(AbstractDocumentContent, StyleSheet)
未能转换 javax.swing.text.html.HTMLDocument.HTMLDocument(StyleSheet)
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.BlockAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.CharacterAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.FormAction

未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.HiddenAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.lsindexAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.ParagraphAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.PreAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.SpecialAction
未能转换 javax.swing.text.html.HTMLDocument.HTMLReader.TagAction
未能转换 javax.swing.text.html.HTMLDocument.insert
未能转换 javax.swing.text.html.HTMLDocument.insertAfterEnd
未能转换 javax.swing.text.html.HTMLDocument.insertAfterStart
未能转换 javax.swing.text.html.HTMLDocument.insertBeforeEnd
未能转换 javax.swing.text.html.HTMLDocument.insertBeforeStart
未能转换 javax.swing.text.html.HTMLDocument.insertUpdate
未能转换 javax.swing.text.html.HTMLDocument.iterator.getAttributes
未能转换 javax.swing.text.html.HTMLDocument.iterator.getEndOffset
未能转换 javax.swing.text.html.HTMLDocument.iterator.getStartOffset
未能转换 javax.swing.text.html.HTMLDocument.iterator.isValid
未能转换 javax.swing.text.html.HTMLDocument.iterator.iterator
未能转换 javax.swing.text.html.HTMLDocument.processHTMLFrameHyperlinkEvent
未能转换 javax.swing.text.html.HTMLDocument.RunElement
未能转换 javax.swing.text.html.HTMLDocument.setInnnerHTML
未能转换 javax.swing.text.html.HTMLDocument.setOuterHTML
未能转换 javax.swing.text.html.HTMLDocument.setParagraphAttributes
未能转换 javax.swing.text.html.HTMLDocument.setParser
未能转换 javax.swing.text.html.HTMLDocument.setPreservesUnknownTags
未能转换 javax.swing.text.html.HTMLDocument.setTokenThreshold
未能转换 javax.swing.text.html.HTMLEditorKit
未能转换 javax.swing.text.html.HTMLEditorKit.HTMLFactory
未能转换 javax.swing.text.html.HTMLEditorKit.HTMLTextAction
未能转换 javax.swing.text.html.HTMLEditorKit.InsertHTMLTextAction
未能转换 javax.swing.text.html.HTMLEditorKit.InsertHTMLTextAction.html
未能转换 javax.swing.text.html.HTMLEditorKit.LinkController
未能转换 javax.swing.text.html.HTMLEditorKit.Parser
未能转换 javax.swing.text.html.HTMLEditorKit.ParserCallback
未能转换 javax.swing.text.html.HTMLFrameHyperlinkEvent
未能转换 javax.swing.text.html.HTMLWriter
未能转换 javax.swing.text.html.HTMLWriter.closeOutUnwantedEmbeddedTags
未能转换 javax.swing.text.html.HTMLWriter.comment
未能转换 javax.swing.text.html.HTMLWriter.emptyTag
未能转换 javax.swing.text.html.HTMLWriter.endTag
未能转换 javax.swing.text.html.HTMLWriter.HTMLWriter

未能转换 javax.swing.text.html.HTMLWriter.isBlockTag
未能转换 javax.swing.text.html.HTMLWriter.matchNameAttribute
未能转换 javax.swing.text.html.HTMLWriter.selectContent
未能转换 javax.swing.text.html.HTMLWriter.startTag
未能转换 javax.swing.text.html.HTMLWriter.synthesizedElement
未能转换 javax.swing.text.html.HTMLWriter.text
未能转换 javax.swing.text.html.HTMLWriter.textAreaContent
未能转换 javax.swing.text.html.HTMLWriter.writeEmbeddedTags
未能转换 javax.swing.text.html.InlineView
未能转换 javax.swing.text.html.ListView
未能转换 javax.swing.text.html.MinimalHTMLWriter.endFontTag
未能转换 javax.swing.text.html.MinimalHTMLWriter.inFontTag
未能转换 javax.swing.text.html.MinimalHTMLWriter.isText
未能转换 javax.swing.text.html.MinimalHTMLWriter.MinimalHTMLWriter
未能转换 javax.swing.text.html.MinimalHTMLWriter.startFontTag
未能转换 javax.swing.text.html.MinimalHTMLWriter.text
未能转换 javax.swing.text.html.MinimalHTMLWriter.write
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeComponent
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeContent
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeHeader
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeHTMLTags
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeImage
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeLeaf
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeNonHTMLAttributes
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeStartParagraph
未能转换 javax.swing.text.html.MinimalHTMLWriter.writeStyles
未能转换 javax.swing.text.html.ObjectView
未能转换 javax.swing.text.html.Option.getAttributes
未能转换 javax.swing.text.html.Option.Option
未能转换 javax.swing.text.html.ParagraphView
未能转换 javax.swing.text.html.parser.AttributeList
未能转换 javax.swing.text.html.parser.ContentModel
未能转换 javax.swing.text.html.parser.DocumentParser
未能转换 javax.swing.text.html.parserDTD
未能转换 javax.swing.text.html.parserDTDConstants
未能转换 javax.swing.text.html.parser.Element
未能转换 javax.swing.text.html.parser.Entity
未能转换 javax.swing.text.html.parser.Parser
未能转换 javax.swing.text.html.parser.ParserDelegator
未能转换 javax.swing.text.html.parser.TagElement

未能转换 javax.swing.text.html.StyleSheet.addAttribute
未能转换 javax.swing.text.html.StyleSheet.addAttribute
未能转换 javax.swing.text.html.StyleSheet.addCSSAttribute
未能转换 javax.swing.text.html.StyleSheet.addCSSAttributeFromHTML
未能转换 javax.swing.text.html.StyleSheet.addStyleSheet
未能转换 javax.swing.text.html.StyleSheet.BoxPainter
未能转换 javax.swing.text.html.StyleSheet.createLargeAttributeSet
未能转换 javax.swing.text.html.StyleSheet.createSmallAttributeSet
未能转换 javax.swing.text.html.StyleSheet.getBackground
未能转换 javax.swing.text.html.StyleSheet.getBase
未能转换 javax.swing.text.html.StyleSheet.getBoxPainter
未能转换 javax.swing.text.html.StyleSheet.getDeclaration
未能转换 javax.swing.text.html.StyleSheet.getFont
未能转换 javax.swing.text.html.StyleSheet.getForeground
未能转换 javax.swing.text.html.StyleSheet.getIndexOfSize
未能转换 javax.swing.text.html.StyleSheet.getListPainter
未能转换 javax.swing.text.html.StyleSheet.getPointSize(int)
未能转换 javax.swing.text.html.StyleSheet.getPointSize(String)
未能转换 javax.swing.text.html.StyleSheet.getRule
未能转换 javax.swing.text.html.StyleSheet.getStyleSheets
未能转换 javax.swing.text.html.StyleSheet.getViewAttributes
未能转换 javax.swing.text.html.StyleSheet.importStyleSheet
未能转换 javax.swing.text.html.StyleSheet.ListPainter
未能转换 javax.swing.text.html.StyleSheet.loadRules
未能转换 javax.swing.text.html.StyleSheet.removeAttribute
未能转换 javax.swing.text.html.StyleSheet.removeAttribute
未能转换 javax.swing.text.html.StyleSheet.removeStyle
未能转换 javax.swing.text.html.StyleSheet.removeStyleSheet
未能转换 javax.swing.text.html.StyleSheet.setBase
未能转换 javax.swing.text.html.StyleSheet.setBaseFontSize(int)
未能转换 javax.swing.text.html.StyleSheet.setBaseFontSize(String)
未能转换 javax.swing.text.html.StyleSheet.stringToColor
未能转换 javax.swing.text.html.StyleSheet.StyleSheet
未能转换 javax.swing.text.html.StyleSheet.translateHTMLToCSS
未能转换 javax.swing.text.IconView
未能转换 javax.swing.text.MutableAttributeSet.addAttribute
未能转换 javax.swing.text.MutableAttributeSet.removeAttribute(AttributeSet)
未能转换 javax.swing.text.MutableAttributeSet.removeAttribute(Enumeration)
未能转换 javax.swing.text.MutableAttributeSet.setResolveParent
未能转换 javax.swing.text.JTextComponent.AccessibleJTextComponent

未能转换 javax.swing.text.JTextComponent.addCaretListener
未能转换 javax.swing.text.JTextComponent.addInputMethodListener
未能转换 javax.swing.text.JTextComponent.addKeymap
未能转换 javax.swing.text.JTextComponent.DEFAULT_KEYMAP
未能转换 javax.swing.text.JTextComponent.fireCaretUpdate
未能转换 javax.swing.text.JTextComponent.FOCUS_ACCELERATOR_KEY
未能转换 javax.swing.text.JTextComponent.getActions
未能转换 javax.swing.text.JTextComponent.getCaret
未能转换 javax.swing.text.JTextComponent.getCaretColor
未能转换 javax.swing.text.JTextComponent.getDisabledTextColor
未能转换 javax.swing.text.JTextComponent.getDocument
未能转换 javax.swing.text.JTextComponent.getFocusAccelerator
未能转换 javax.swing.text.JTextComponent.getHighlighter
未能转换 javax.swing.text.JTextComponent.getInputMethodRequests
未能转换 javax.swing.text.JTextComponent.getKeymap
未能转换 javax.swing.text.JTextComponent.getKeymap(String)
未能转换 javax.swing.text.JTextComponent.getMargin
未能转换 javax.swing.text.JTextComponent.getPreferredSize
未能转换 javax.swing.text.JTextComponent.getScrollableBlockIncrement
未能转换 javax.swing.text.JTextComponent.getScrollableTracksViewportHeight
未能转换 javax.swing.text.JTextComponent.getScrollableTracksViewportWidth
未能转换 javax.swing.text.JTextComponent.getScrolloableUnitIncrement
未能转换 javax.swing.text.JTextComponent.getSelectedTextColor
未能转换 javax.swing.text.JTextComponent.getSelectionColor
未能转换 javax.swing.text.JTextComponent.getUI
未能转换 javax.swing.text.JTextComponent.loadKeymap
未能转换 javax.swing.text.JTextComponent.modelToView
未能转换 javax.swing.text.JTextComponent.processInputMethodEvent
未能转换 javax.swing.text.JTextComponent.processKeyEvent
未能转换 javax.swing.text.JTextComponent.read
未能转换 javax.swing.text.JTextComponent.removeCaretListener
未能转换 javax.swing.text.JTextComponent.removeKeymap
未能转换 javax.swing.text.JTextComponent.setCaret
未能转换 javax.swing.text.JTextComponent.setCaretColor
未能转换 javax.swing.text.JTextComponent.setDisabledTextColor
未能转换 javax.swing.text.JTextComponent.setDocument
未能转换 javax.swing.text.JTextComponent.setFocusAccelerator
未能转换 javax.swing.text.JTextComponent.setHighlighter
未能转换 javax.swing.text.JTextComponent.setKeymap
未能转换 javax.swing.text.JTextComponent.setMargin

未能转换 javax.swing.text.JTextComponent.setSelectedTextColor
未能转换 javax.swing.text.JTextComponent.setSelectionColor
未能转换 javax.swing.text.JTextComponent.setText
未能转换 javax.swing.text.JTextComponent.setUI
未能转换 javax.swing.text.JTextComponent.updateUI
未能转换 javax.swing.text.JTextComponent.viewToModel
未能转换 javax.swing.text.JTextComponent.write
未能转换 javax.swing.text.JTextComponentBeanInfo
未能转换 javax.swing.text.Keymap
未能转换 javax.swing.text.LabelView
未能转换 javax.swing.text.LayeredHighlighter
未能转换 javax.swing.text.LayeredHighlighter.LayerPainter
未能转换 javax.swing.text.LayoutQueue
未能转换 javax.swing.text.MutableAttributeSet.addAttribute
未能转换 javax.swing.text.MutableAttributeSet.removeAttribute(AttributeSet)
未能转换 javax.swing.text.MutableAttributeSet.removeAttribute(Enumeration)
未能转换 javax.swing.text.MutableAttributeSet.setResolveParent
未能转换 javax.swing.text.ParagraphView
未能转换 javax.swing.text.PasswordView
未能转换 javax.swing.text.PlainDocument
未能转换 javax.swing.text.PlainView
未能转换 javax.swing.text.Position
未能转换 javax.swing.text.Position.Bias
未能转换 javax.swing.text.rtf.RTFFormatKit
未能转换 javax.swing.text.SimpleAttributeSet.addAttribute
未能转换 javax.swing.text.SimpleAttributeSet.containsAttributes
未能转换 javax.swing.text.SimpleAttributeSet.copyAttributes
未能转换 javax.swing.text.SimpleAttributeSet.EMPTY
未能转换 javax.swing.text.SimpleAttributeSet.equals
未能转换 javax.swing.text.SimpleAttributeSet.getAttributeNames
未能转换 javax.swing.text.SimpleAttributeSet.getResolveParent
未能转换 javax.swing.text.SimpleAttributeSet.isEqual
未能转换 javax.swing.text.SimpleAttributeSet.removeAttribute(AttributeSet)
未能转换 javax.swing.text.SimpleAttributeSet.removeAttribute(Enumeration)
未能转换 javax.swing.text.SimpleAttributeSet.setResolveParent
未能转换 javax.swing.text.SimpleAttributeSet.toString
未能转换 javax.swing.text.StringContent.createPosition
未能转换 javax.swing.text.StringContent.getPositionsInRange
未能转换 javax.swing.text.StringContent.updateUndoPositions
未能转换 javax.swing.text.Style

未能转换 javax.swing.text.StyleConstants
未能转换 javax.swing.text.StyleConstants.CharacterConstants
未能转换 javax.swing.text.StyleConstants.ColorConstants
未能转换 javax.swing.text.StyleConstants.FontConstants
未能转换 javax.swing.text.StyleConstants.ParagraphConstants
未能转换 javax.swing.text.StyleContext
未能转换 javax.swing.text.StyleContext.NamedStyle
未能转换 javax.swing.text.StyleContext.SmallAttributeSet
未能转换 javax.swing.text.StyledDocument
未能转换 javax.swing.text.StyledEditorKit
未能转换 javax.swing.text.StyledEditorKit.AlignmentAction
未能转换 javax.swing.text.StyledEditorKit.BoldAction
未能转换 javax.swing.text.StyledEditorKit.FontFamilyAction
未能转换 javax.swing.text.StyledEditorKit.FontSizeAction
未能转换 javax.swing.text.StyledEditorKit.ForegroundAction
未能转换 javax.swing.text.StyledEditorKit.ItalicAction
未能转换 javax.swing.text.StyledEditorKit.StyledTextAction
未能转换 javax.swing.text.StyledEditorKit.UnderlineAction
未能转换 javax.swing.text.TabableView
未能转换 javax.swing.text.TabExpander
未能转换 javax.swing.text.TableView
未能转换 javax.swing.text.TableView.TableCell
未能转换 javax.swing.text.TableView.TableRow
未能转换 javax.swing.text.TabSet
未能转换 javax.swing.text.TabStop
未能转换 javax.swing.text.TextAction
未能转换 javax.swing.text.Utilities
未能转换 javax.swing.text.View
未能转换 javax.swing.text.ViewFactory
未能转换 javax.swing.text.WrappedPlainView
未能转换 javax.swing.text.ZoneView

Javax.swing.tree 错误信息

未能转换 javax.swing.tree.AbstractLayoutCache
未能转换 javax.swing.tree.AbstractLayoutCache.NodeDimensions
未能转换 javax.swing.tree.DefaultMutableTreeNode.allowsChildren
未能转换 javax.swing.tree.DefaultMutableTreeNode.breadthFirstEnumeration
未能转换 javax.swing.tree.DefaultMutableTreeNode.children
未能转换 javax.swing.tree.DefaultMutableTreeNode.depthFirstEnumeration
未能转换 javax.swing.tree.DefaultMutableTreeNode.getAllowsChildren
未能转换 javax.swing.tree.DefaultMutableTreeNode.getDepth
未能转换 javax.swing.tree.DefaultMutableTreeNode.getLeafCount
未能转换 javax.swing.tree.DefaultMutableTreeNode.getPath
未能转换 javax.swing.tree.DefaultMutableTreeNode.getUserObject
未能转换 javax.swing.tree.DefaultMutableTreeNode.getUserObjectPath
未能转换 javax.swing.tree.DefaultMutableTreeNode.isNodeDescendant
未能转换 javax.swing.tree.DefaultMutableTreeNode.setAllowsChildren
未能转换 javax.swing.tree.DefaultMutableTreeNode.setUserObject
未能转换 javax.swing.tree.DefaultMutableTreeNode.userObject
未能转换 javax.swing.tree.DefaultTreeCellEditor
未能转换 javax.swing.tree.DefaultTreeCellEditor.DefaultTextField
未能转换 javax.swing.tree.DefaultTreeCellEditor.EditorContainer
未能转换 javax.swing.tree.DefaultTreeCellRenderer
未能转换 javax.swing.tree.DefaultTreeModel
未能转换 javax.swing.tree.DefaultTreeModel.addTreeModelListener
未能转换 javax.swing.tree.DefaultTreeModel.asksAllowsChildren
未能转换 javax.swing.tree.DefaultTreeModel.DefaultTreeModel(TreeNode)
未能转换 javax.swing.tree.DefaultTreeModel.DefaultTreeModel(TreeNode, boolean)
未能转换 javax.swing.tree.DefaultTreeModel.fireTreeNodesChanged
未能转换 javax.swing.tree.DefaultTreeModel.fireTreeNodesInserted
未能转换 javax.swing.tree.DefaultTreeModel.fireTreeNodesRemoved
未能转换 javax.swing.tree.DefaultTreeModel.fireTreeStructureChanged
未能转换 javax.swing.tree.DefaultTreeModel.getChild
未能转换 javax.swing.tree.DefaultTreeModel getChildCount
未能转换 javax.swing.tree.DefaultTreeModel.getIndexofChild
未能转换 javax.swing.tree.DefaultTreeModel.getListeners
未能转换 javax.swing.tree.DefaultTreeModel.getRoot
未能转换 javax.swing.tree.DefaultTreeModel.insertNodeinto
未能转换 javax.swing.tree.DefaultTreeModel.isLeaf
未能转换 javax.swing.tree.DefaultTreeModel.listenerList

未能转换 javax.swing.tree.DefaultTreeModel.nodeChanged
未能转换 javax.swing.tree.DefaultTreeModel.nodesChanged
未能转换 javax.swing.tree.DefaultTreeModel.nodeStructureChanged
未能转换 javax.swing.tree.DefaultTreeModel.nodesWereInserted
未能转换 javax.swing.tree.DefaultTreeModel.nodesWereRemoved
未能转换 javax.swing.tree.DefaultTreeModel.reload
未能转换 javax.swing.tree.DefaultTreeModel.removeNodeFromParent
未能转换 javax.swing.tree.DefaultTreeModel.removeTreeModelListener
未能转换 javax.swing.tree.DefaultTreeModel.root
未能转换 javax.swing.tree.DefaultTreeModel.setAsksAllowsChildren
未能转换 javax.swing.tree.DefaultTreeModel.setRoot
未能转换 javax.swing.tree.DefaultTreeModel.valueForPathChanged
未能转换 javax.swing.tree.DefaultTreeSelectionModel
未能转换 javax.swing.tree.ExpandVetoException.event
未能转换 javax.swing.tree.ExpandVetoException.ExpandVetoException(TreeExpansionEvent)
未能转换 javax.swing.tree.ExpandVetoException.ExpandVetoException(TreeExpansionEvent, String)
未能转换 javax.swing.tree.FixedHeightLayoutCache
未能转换 javax.swing.tree.MutableTreeNode.setUserObject
未能转换 javax.swing.tree.RowMapper
未能转换 javax.swing.tree.TreeCellEditor
未能转换 javax.swing.tree.TreeCellRenderer
未能转换 javax.swing.tree.TreeModel
未能转换 javax.swing.tree.TreeModel.addTreeModelListener
未能转换 javax.swing.tree.TreeModel.getChild
未能转换 javax.swing.tree.TreeModel getChildCount
未能转换 javax.swing.tree.TreeModel getIndexofChild
未能转换 javax.swing.tree.TreeModel.getRoot
未能转换 javax.swing.tree.TreeModel.isLeaf
未能转换 javax.swing.tree.TreeModel.removeTreeModelListener
未能转换 javax.swing.tree.TreeModel.valueForPathChanged
未能转换 javax.swing.treeTreeNode.getAllowsChildren
未能转换 javax.swing.tree.TreePath
未能转换 javax.swing.tree.TreeSelectionModel
未能转换 javax.swing.tree.VariableHeightLayoutCache

Javax.swing.undo 错误信息

未能转换 javax.swing.undo.AbstractUndoableEdit

未能转换 javax.swing.undo.CompoundEdit

未能转换 javax.swing.undo.StateEdit

未能转换 javax.swing.undo.StateEditable

未能转换 javax.swing.undo.UndoableEdit

未能转换 javax.swing.undo.UndoableEditSupport

未能转换 javax.swing.undo.UndoManager

Java.transaction 错误信息

未能转换 javax.transaction.HeuristicCommitException
未能转换 javax.transaction.HeuristicMixedException
未能转换 javax.transaction.HeuristicRollbackException
未能转换 javax.transaction.InvalidTransactionException
未能转换 javax.transaction.RollbackException
未能转换 javax.transaction.Status.STATUS_ACTIVE
未能转换 javax.transaction.Status.STATUS_COMMITTING
未能转换 javax.transaction.Status.STATUS_MARKED_ROLLBACK
未能转换 javax.transaction.Status.STATUS_PREPARED
未能转换 javax.transaction.Status.STATUS_PREPARING
未能转换 javax.transaction.Status.STATUS_UNKNOWN
未能转换 javax.transaction.SystemException.errorCode
未能转换 javax.transaction.SystemException.SystemException
未能转换 javax.transaction.Synchronization
未能转换 javax.transaction.Transaction
未能转换 javax.transaction.Transaction.commit
未能转换 javax.transaction.Transaction.delistResource
未能转换 javax.transaction.Transaction.enlistResource
未能转换 javax.transaction.Transaction.getStatus
未能转换 javax.transaction.Transaction.registerSynchronization
未能转换 javax.transaction.Transaction.rollback
未能转换 javax.transaction.Transaction.setRollbackOnly
未能转换 javax.transaction.TransactionManager
未能转换 javax.transaction.TransactionManager.setRollbackOnly
未能转换 javax.transaction.TransactionRequiredException
未能转换 javax.transaction.TransactionRolledbackException
未能转换 javax.transaction.UserTransaction
未能转换 javax.transaction.UserTransaction.begin
未能转换 javax.transaction.UserTransaction.setRollbackOnly
未能转换 javax.transaction.xa.XAException
未能转换 javax.transaction.xa.XAResource
未能转换 javax.transaction.xa.Xid

Javax.xml 错误信息

未能转换 javax.xml.parsers.DocumentBuilder
未能转换 javax.xml.parsers.DocumentBuilder.isValidating
未能转换 javax.xml.parsers.DocumentBuilder.parse
未能转换 javax.xml.parsers.DocumentBuilder.setEntityResolver
未能转换 javax.xml.parsers.DocumentBuilder.setErrorHandler
未能转换 javax.xml.parsers.DocumentBuilderFactory
未能转换 javax.xml.parsers.FactoryConfigurationError
未能转换 javax.xml.parsers.ParserConfigurationException
未能转换 javax.xml.parsers.SAXParser.getProperty
未能转换 javax.xml.parsers.SAXParser.setProperty
未能转换 javax.xml.parsers.SAXParserFactory.getFeature
未能转换 javax.xml.parsers.SAXParserFactory.setFeature
未能转换 javax.xml.transform.dom.DOMLocator
未能转换 javax.xml.transform.dom.DOMResult
未能转换 javax.xml.transform.dom.DOMResult.DOMResult
未能转换 javax.xml.transform.dom.DOMSource
未能转换 javax.xml.transform.ErrorListener
未能转换 javax.xml.transform.OutputKeys
未能转换 javax.xml.transform.Result.PI_DISABLE_OUTPUT_ESCAPING
未能转换 javax.xml.transform.Result.PI_ENABLE_OUTPUT_ESCAPING
未能转换 javax.xml.transform.sax.SAXResult
未能转换 javax.xml.transform.sax.SAXResult.SAXResult
未能转换 javax.xml.transform.sax.SAXTransformerFactory
未能转换 javax.xml.transform.sax.TemplatesHandler
未能转换 javax.xml.transform.sax.TransformerHandler
未能转换 javax.xml.transform.stream.StreamResult.setSystemId
未能转换 javax.xml.transform.stream.StreamSource.getPublicId
未能转换 javax.xml.transform.stream.StreamSource.setPublicId
未能转换 javax.xml.transform.stream.StreamSource.setSystemId
未能转换 javax.xml.transform.Templates.getOutputProperties
未能转换 javax.xml.transform.Transformer.getOutputProperties
未能转换 javax.xml.transform.Transformer.getProperty
未能转换 javax.xml.transform.Transformer.getURIResolver
未能转换 javax.xml.transform.Transformer.setErrorListener
未能转换 javax.xml.transform.Transformer.setOutputProperties
未能转换 javax.xml.transform.Transformer.setOutputProperty
未能转换 javax.xml.transform.Transformer.transform

未能转换 javax.xml.transform.TransformerConfigurationException.TransformerConfigurationException
未能转换 javax.xml.transform.TransformerException.initCause
未能转换 javax.xml.transform.TransformerException.setLocator
未能转换 javax.xml.transform.TransformerException.TransformerException
未能转换 javax.xml.transform.TransformerFactory.getAssociatedStylesheet
未能转换 javax.xml.transform.TransformerFactory.getAttribute
未能转换 javax.xml.transform.TransformerFactory.getURIResolver
未能转换 javax.xml.transform.TransformerFactory.newTransformer
未能转换 javax.xml.transform.TransformerFactory.setAttribute
未能转换 javax.xml.transform.TransformerFactory.setErrorListener
未能转换 javax.xml.transform.TransformerFactoryConfigurationError
未能转换 javax.xml.transform.URIResolver
未能转换 javax.xml.transform.URIResolver.resolve

Org.omg 错误信息

未能转换 org.omg.CORBA._IDLTypeStub
未能转换 org.omg.CORBA._PolicyStub
未能转换 org.omg.CORBA.Any.create_input_stream
未能转换 org.omg.CORBA.Any.create_output_stream
未能转换 org.omg.CORBA.Any.extract_Principal
未能转换 org.omg.CORBA.Any.extract_Value
未能转换 org.omg.CORBA.Any.insert_fixed
未能转换 org.omg.CORBA.Any.insert_Object
未能转换 org.omg.CORBA.Any.insert_Principal
未能转换 org.omg.CORBA.Any.insert_Streamable
未能转换 org.omg.CORBA.Any.insert_Value(Serializable)
未能转换 org.omg.CORBA.Any.insert_Value(Serializable, TypeCode)
未能转换 org.omg.CORBA.Any.read_value
未能转换 org.omg.CORBA.Any.type
未能转换 org.omg.CORBA.Any.write_value
未能转换 org.omg.CORBA.AnyHolder
未能转换 org.omg.CORBA.AnyHolder._read
未能转换 org.omg.CORBA.AnyHolder._write
未能转换 org.omg.CORBA.AnySeqHelper
未能转换 org.omg.CORBA.AnySeqHolder
未能转换 org.omg.CORBA.ARG_IN
未能转换 org.omg.CORBA.ARG_INOUT
未能转换 org.omg.CORBA.ARG_OUT
未能转换 org.omg.CORBA.BAD_CONTEXT.BAD_CONTEXT(int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_CONTEXT.BAD_CONTEXT(String, int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_INV_ORDER.BAD_INV_ORDER(int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_INV_ORDER.BAD_INV_ORDER(String, int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_OPERATION.BAD_OPERATION(int, ComletionStatus)
未能转换 org.omg.CORBA.BAD_OPERATION.BAD_OPERATION(String, int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_PARAM.BAD_PARAM(int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_PARAM.BAD_PARAM(String, int, CompletionStatus)
未能转换 org.omg.CORBA.BAD_POLICY
未能转换 org.omg.CORBA.BAD_POLICY_TYPE
未能转换 org.omg.CORBA.BAD_POLICY_VALUE
未能转换 org.omg.CORBA.BooleanHolder._read
未能转换 org.omg.CORBA.BooleanHolder._write
未能转换 org.omg.CORBA.BooleanSeqHelper

未能转换 org.omg.CORBA.BooleanSeqHolder
未能转换 org.omg.CORBA.ByteHolder._read
未能转换 org.omg.CORBA.ByteHolder._write
未能转换 org.omg.CORBA.CharHolder._read
未能转换 org.omg.CORBA.CharHolder._write
未能转换 org.omg.CORBA.CharSeqHelper
未能转换 org.omg.CORBA.CharSeqHolder
未能转换 org.omg.CORBA.CompletionStatus
未能转换 org.omg.CORBA.CompletionStatusHelper
未能转换 org.omg.CORBA.Context
未能转换 org.omg.CORBA.ContextList
未能转换 org.omg.CORBA.CTX_RESTRICT_SCOPE
未能转换 org.omg.CORBA.Current
未能转换 org.omg.CORBA.CurrentHelper
未能转换 org.omg.CORBA.CurrentHolder
未能转换 org.omg.CORBA.CurrentOperations
未能转换 org.omg.CORBA.CustomMarshal
未能转换 org.omg.CORBA.DataInputStream
未能转换 org.omg.CORBA.DataOutputStream
未能转换 org.omg.CORBA.DefinitionKind
未能转换 org.omg.CORBA.DefinitionKindHelper
未能转换 org.omg.CORBA.DomainManager
未能转换 org.omg.CORBA.DomainManagerOperations
未能转换 org.omg.CORBA.DoubleHolder._read
未能转换 org.omg.CORBA.DoubleHolder._write
未能转换 org.omg.CORBA.DoubleSeqHelper
未能转换 org.omg.CORBA.DoubleSeqHolder
未能转换 org.omg.CORBA.DynamicImplementation
未能转换 org.omg.CORBA.DynAny
未能转换 org.omg.CORBA.DynArray
未能转换 org.omg.CORBA.DynEnum
未能转换 org.omg.CORBA.DynFixed
未能转换 org.omg.CORBA.DynSequence
未能转换 org.omg.CORBA.DynStruct
未能转换 org.omg.CORBA.DynUnion
未能转换 org.omg.CORBA.DynValue
未能转换 org.omg.CORBA.Environment
未能转换 org.omg.CORBA.ExceptionList
未能转换 org.omg.CORBA.FieldNameHelper
未能转换 org.omg.CORBA.FixedHolder

未能转换 org.omg.CORBA.FloatHolder._read
未能转换 org.omg.CORBA.FloatHolder._write
未能转换 org.omg.CORBA.FloatSeqHelper
未能转换 org.omg.CORBA.FloatSeqHolder
未能转换 org.omg.CORBA.IdentifierHelper
未能转换 org.omg.CORBA.IDLType
未能转换 org.omg.CORBA.IDLTypeHelper
未能转换 org.omg.CORBA.IDLTypeOperations
未能转换 org.omg.CORBA.IntHolder._read
未能转换 org.omg.CORBA.IntHolder._write
未能转换 org.omg.CORBA.INV_POLICY.INV_POLICY(int, CompletionStatus)
未能转换 org.omg.CORBA.INV_POLICY.INV_POLICY(String, int, CompletionStatus)
未能转换 org.omg.CORBA.IRObject
未能转换 org.omg.CORBA.IRObjectOperations
未能转换 org.omg.CORBA.LocalObject
未能转换 org.omg.CORBA.LongHolder._read
未能转换 org.omg.CORBA.LongHolder._write
未能转换 org.omg.CORBA.LongLongSeqHelper
未能转换 org.omg.CORBA.LongLongSeqHolder
未能转换 org.omg.CORBA.LongSeqHelper
未能转换 org.omg.CORBA.LongSeqHolder
未能转换 org.omg.CORBA.MARSHAL.MARSHAL(int, CompletionStatus)
未能转换 org.omg.CORBA.MARSHAL.MARSHAL(String, int, CompletionStatus)
未能转换 org.omg.CORBA.NamedValue
未能转换 org.omg.CORBA.NameValuePair
未能转换 org.omg.CORBA.NameValuePairHelper
未能转换 org.omg.CORBA.NO_PERMISSION.NO_PERMISSION(int, CompletionStatus)
未能转换 org.omg.CORBA.NO_PERMISSION.NO_PERMISSION(String, int, CompletionStatus)
未能转换 org.omg.CORBA.NVList
未能转换 org.omg.CORBA.Object._create_request(Context, String, NVList, NamedValue)
未能转换 org.omg.CORBA.Object._create_request(Context, String, NVList, NamedValue, ExceptionList, ContextList)
未能转换 org.omg.CORBA.Object._duplicate
未能转换 org.omg.CORBA.Object._get_domain_managers
未能转换 org.omg.CORBA.Object._get_interface_def
未能转换 org.omg.CORBA.Object._get_policy
未能转换 org.omg.CORBA.Object._hash
未能转换 org.omg.CORBA.Object._is_a
未能转换 org.omg.CORBA.Object._is_equivalent
未能转换 org.omg.CORBA.Object._non_existent
未能转换 org.omg.CORBA.Object._release

未能转换 org.omg.CORBA.Object._request
未能转换 org.omg.CORBA.Object._set_policy_override
未能转换 org.omg.CORBA.ObjectHelper
未能转换 org.omg.CORBA.ObjectHolder._read
未能转换 org.omg.CORBA.ObjectHolder._write
未能转换 org.omg.CORBA.OctetSeqHelper
未能转换 org.omg.CORBA.OctetSeqHolder
未能转换 org.omg.CORBA.OMGVMCID
未能转换 org.omg.CORBA.ORB
未能转换 org.omg.CORBA.Policy
未能转换 org.omg.CORBA.PolicyError.reason
未能转换 org.omg.CORBA.PolicyHelper
未能转换 org.omg.CORBA.PolicyHolder
未能转换 org.omg.CORBA.PolicyListHelper
未能转换 org.omg.CORBA.PolicyListHolder
未能转换 org.omg.CORBA.PolicyOperations
未能转换 org.omg.CORBA.PolicyTypeHelper
未能转换 org.omg.CORBA.portable.ApplicationException.getIld
未能转换 org.omg.CORBA.portable.ApplicationException.getInputStream
未能转换 org.omg.CORBA.portable.BoxedValueHelper
未能转换 org.omg.CORBA.portable.CustomValue
未能转换 org.omg.CORBA.portable.Delegate
未能转换 org.omg.CORBA.portable.IDLEntity
未能转换 org.omg.CORBA.portable.IndirectionException.IndirectionException
未能转换 org.omg.CORBA.portable.IndirectionException.offset
未能转换 org.omg.CORBA.portable.InputStream
未能转换 org.omg.CORBA.portable.InvokeHandler
未能转换 org.omg.CORBA.portable.ObjectImpl._create_request(Context, String, NVList, NamedValue)
未能转换 org.omg.CORBA.portable.ObjectImpl._create_request(Context, String, NVList, NamedValue, ExceptionList, ContextList)
未能转换 org.omg.CORBA.portable.ObjectImpl._duplicate
未能转换 org.omg.CORBA.portable.ObjectImpl._get_delegate
未能转换 org.omg.CORBA.portable.ObjectImpl._get_domain_managers
未能转换 org.omg.CORBA.portable.ObjectImpl._get_interface_def
未能转换 org.omg.CORBA.portable.ObjectImpl._get_policy
未能转换 org.omg.CORBA.portable.ObjectImpl._hash
未能转换 org.omg.CORBA.portable.ObjectImpl._ids
未能转换 org.omg.CORBA.portable.ObjectImpl._invoke
未能转换 org.omg.CORBA.portable.ObjectImpl._is_a
未能转换 org.omg.CORBA.portable.ObjectImpl._is_equivalent
未能转换 org.omg.CORBA.portable.ObjectImpl._is_local

未能转换 org.omg.CORBA.portable.ObjectImpl._non_existent
未能转换 org.omg.CORBA.portable.ObjectImpl._orb
未能转换 org.omg.CORBA.portable.ObjectImpl._release
未能转换 org.omg.CORBA.portable.ObjectImpl._releaseReply
未能转换 org.omg.CORBA.portable.ObjectImpl._request(String)
未能转换 org.omg.CORBA.portable.ObjectImpl._request(String, boolean)
未能转换 org.omg.CORBA.portable.ObjectImpl._servant_postinvoke
未能转换 org.omg.CORBA.portable.ObjectImpl._servant_preinvoke
未能转换 org.omg.CORBA.portable.ObjectImpl._set_delegate
未能转换 org.omg.CORBA.portable.ObjectImpl._set_policy_override
未能转换 org.omg.CORBA.portable.OutputStream
未能转换 org.omg.CORBA.portable.ResponseHandler
未能转换 org.omg.CORBA.portable.ServantObject
未能转换 org.omg.CORBA.portable.Streamable
未能转换 org.omg.CORBA.portable.StreamableValue
未能转换 org.omg.CORBA.portable.ValueBase
未能转换 org.omg.CORBA.portable.ValueFactory
未能转换 org.omg.CORBA.Principal
未能转换 org.omg.CORBA.PrincipalHolder
未能转换 org.omg.CORBA.PRIVATE_MEMBER
未能转换 org.omg.CORBA.PUBLIC_MEMBER
未能转换 org.omg.CORBA.RepositoryIdHelper
未能转换 org.omg.CORBA.Request
未能转换 org.omg.CORBA.ServerRequest
未能转换 org.omg.CORBA.ServiceDetail
未能转换 org.omg.CORBA.ServiceDetailHelper
未能转换 org.omg.CORBA.ServiceInformation
未能转换 org.omg.CORBA.ServiceInformationHelper
未能转换 org.omg.CORBA.ServiceInformationHolder
未能转换 org.omg.CORBA.SetOverrideType
未能转换 org.omg.CORBA.SetOverrideTypeHelper
未能转换 org.omg.CORBA.ShortHolder._read
未能转换 org.omg.CORBA.ShortHolder._write
未能转换 org.omg.CORBA.ShortSeqHelper
未能转换 org.omg.CORBA.ShortSeqHolder
未能转换 org.omg.CORBA.StringHolder._read
未能转换 org.omg.CORBA.StringHolder._write
未能转换 org.omg.CORBA.StringHolder.StringHolder
未能转换 org.omg.CORBA.StringHolder.StringHolder(String)
未能转换 org.omg.CORBA.StringValueHelper

未能转换 org.omg.CORBA.StructMember
未能转换 org.omg.CORBA.StructMemberHelper
未能转换 org.omg.CORBA.SystemException.completed
未能转换 org.omg.CORBA.SystemException.minor
未能转换 org.omg.CORBA.TCKind.<DataType>
未能转换 org.omg.CORBA.TCKind.TCKind
未能转换 org.omg.CORBA.TypeCode
未能转换 org.omg.CORBA.TypeCode.concrete_base_type
未能转换 org.omg.CORBA.TypeCode.content_type
未能转换 org.omg.CORBA.TypeCode.default_index
未能转换 org.omg.CORBA.TypeCode.discriminator_type
未能转换 org.omg.CORBA.TypeCode.equivalent
未能转换 org.omg.CORBA.TypeCode.fixed_digits
未能转换 org.omg.CORBA.TypeCode.fixed_scale
未能转换 org.omg.CORBA.TypeCode.get_compact_typecode
未能转换 org.omg.CORBA.TypeCode.id
未能转换 org.omg.CORBA.TypeCode.length
未能转换 org.omg.CORBA.TypeCode.member_count
未能转换 org.omg.CORBA.TypeCode.member_label
未能转换 org.omg.CORBA.TypeCode.member_name
未能转换 org.omg.CORBA.TypeCode.member_type
未能转换 org.omg.CORBA.TypeCode.member_visibility
未能转换 org.omg.CORBA.TypeCode.name
未能转换 org.omg.CORBA.TypeCode.type_modifier
未能转换 org.omg.CORBA.TypeCode.TypeCode
未能转换 org.omg.CORBA.TypeCodeHolder
未能转换 org.omg.CORBA.ULongLongSeqHelper
未能转换 org.omg.CORBA.ULongLongSeqHolder
未能转换 org.omg.CORBA.ULongSeqHelper
未能转换 org.omg.CORBA.ULongSeqHolder
未能转换 org.omg.CORBA.UnionMember
未能转换 org.omg.CORBA.UnionMemberHelper
未能转换 org.omg.CORBA.UnknownUserException.except
未能转换 org.omg.CORBA.UNSUPPORTED_POLICY
未能转换 org.omg.CORBA.UNSUPPORTED_POLICY_VALUE
未能转换 org.omg.CORBA.UShortSeqHelper
未能转换 org.omg.CORBA.UShortSeqHolder
未能转换 org.omg.CORBA.ValueBaseHelper
未能转换 org.omg.CORBA.ValueBaseHolder
未能转换 org.omg.CORBA.ValueBaseHolder._read

未能转换 org.omg.CORBA.ValueBaseHolder._write
未能转换 org.omg.CORBA.ValueBaseHolder.ValueBaseHolder
未能转换 org.omg.CORBA.ValueMember
未能转换 org.omg.CORBA.ValueMemberHelper
未能转换 org.omg.CORBA.VersionSpecHelper
未能转换 org.omg.CORBA.VisibilityHelper
未能转换 org.omg.CORBA.VM_ABSTRACT
未能转换 org.omg.CORBA.VM_CUSTOM
未能转换 org.omg.CORBA.VM_NONE
未能转换 org.omg.CORBA.VM_TRUNCATABLE
未能转换 org.omg.CORBA.WCharSeqHelper
未能转换 org.omg.CORBA.WCharSeqHolder
未能转换 org.omg.CORBA.WStringValueHelper
未能转换 org.omg.CORBA_2_3.ORB
未能转换 org.omg.CORBA_2_3.portable.Delegate
未能转换 org.omg.CORBA_2_3.portable.InputStream
未能转换 org.omg.CORBA_2_3.portable.ObjectImpl
未能转换 org.omg.CORBA_2_3.portable.OutputStream
未能转换 org.omg.CosNaming._BindingIteratorImplBase
未能转换 org.omg.CosNaming._BindingIteratorStub
未能转换 org.omg.CosNaming._NamingContextImplBase
未能转换 org.omg.CosNaming._NamingContextStub
未能转换 org.omg.CosNaming.Binding
未能转换 org.omg.CosNaming.BindingHelper
未能转换 org.omg.CosNaming.BindingHolder
未能转换 org.omg.CosNaming.BindingIterator
未能转换 org.omg.CosNaming.BindingIteratorHelper
未能转换 org.omg.CosNaming.BindingIteratorHolder
未能转换 org.omg.CosNaming.BindingIteratorOperations
未能转换 org.omg.CosNaming.BindingListHelper
未能转换 org.omg.CosNaming.BindingListHolder
未能转换 org.omg.CosNaming.BindingType
未能转换 org.omg.CosNaming.BindingTypeHelper
未能转换 org.omg.CosNaming.BindingTypeHolder
未能转换 org.omg.CosNaming.IStringHelper
未能转换 org.omg.CosNaming.NameComponent
未能转换 org.omg.CosNaming.NameComponentHelper
未能转换 org.omg.CosNaming.NameComponentHolder
未能转换 org.omg.CosNaming.NameHelper
未能转换 org.omg.CosNaming.NameHolder

未能转换 org.omg.CosNaming.NamingContext
未能转换 org.omg.CosNaming.NamingContextHelper
未能转换 org.omg.CosNaming.NamingContextHolder
未能转换 org.omg.CosNaming.NamingContextOperations
未能转换 org.omg.CosNaming.NamingContextPackage.AlreadyBoundHelper
未能转换 org.omg.CosNaming.NamingContextPackage.AlreadyBoundHolder
未能转换 org.omg.CosNaming.NamingContextPackage.CannotProceed.CannotProceed
未能转换 org.omg.CosNaming.NamingContextPackage.CannotProceed.cxt
未能转换 org.omg.CosNaming.NamingContextPackage.CannotProceed.rest_of_name
未能转换 org.omg.CosNaming.NamingContextPackage.CannotProceedHelper
未能转换 org.omg.CosNaming.NamingContextPackage.CannotProceedHolder
未能转换 org.omg.CosNaming.NamingContextPackage.InvalidNameHelper
未能转换 org.omg.CosNaming.NamingContextPackage.InvalidNameHolder
未能转换 org.omg.CosNaming.NamingContextPackage.NotEmptyHelper
未能转换 org.omg.CosNaming.NamingContextPackage.NotEmptyHolder
未能转换 org.omg.CosNaming.NamingContextPackage.NotFound.NotFound
未能转换 org.omg.CosNaming.NamingContextPackage.NotFound.rest_of_name
未能转换 org.omg.CosNaming.NamingContextPackage.NotFound.why
未能转换 org.omg.CosNaming.NamingContextPackage.NotFoundHelper
未能转换 org.omg.CosNaming.NamingContextPackage.NotFoundHolder
未能转换 org.omg.CosNaming.NamingContextPackage.NotFoundReason
未能转换 org.omg.CosNaming.NamingContextPackage.NotFoundReasonHelper
未能转换 org.omg.CosNaming.NamingContextPackage.NotFoundReasonHolder
未能转换 org.omg.stub.java.rmi._Remote_Stub

Org.w3c 错误信息

未能转换 org.w3c.dom.DOMException
未能转换 org.w3c.dom.DOMException.<ExceptionType>
未能转换 org.w3c.dom.DOMException.code
未能转换 org.w3c.dom.DOMImplementation.createDocumentType
未能转换 org.w3c.dom.Element.setAttribute
未能转换 org.w3c.dom.Node.getChildNodes
未能转换 org.w3c.dom.Node.setPrefix
未能转换 org.w3c.dom.ranges.DocumentRange
未能转换 org.w3c.dom.ranges.Range
未能转换 org.w3c.dom.ranges.RangeException
未能转换 org.w3c.dom.traversal.DocumentTraversal.createNodeIterator
未能转换 org.w3c.dom.traversal.DocumentTraversal.createTreeWalker
未能转换 org.w3c.dom.traversal.NodeFilter.SHOW_ALL
未能转换 org.w3c.dom.traversal.NodeIterator.getExpandEntityReferences
未能转换 org.w3c.dom.traversal.TreeWalker.getExpandEntityReferences

Org.xml 错误信息

未能转换 org.xml.sax.AttributeList.getLength
未能转换 org.xml.sax.AttributeList.getName
未能转换 org.xml.sax.AttributeList.getType
未能转换 org.xml.sax.AttributeList.getValue
未能转换 org.xml.sax.Attributes.getIndex
未能转换 org.xml.sax.Attributes.getLength
未能转换 org.xml.sax.Attributes.getLocalName
未能转换 org.xml.sax.Attributes.getQName
未能转换 org.xml.sax.Attributes.getType
未能转换 org.xml.sax.Attributes.getURI
未能转换 org.xml.sax.Attributes.getValue
未能转换 org.xml.sax.DocumentHandler
未能转换 org.xml.sax.DTDHandler
未能转换 org.xml.sax.EntityResolver
未能转换 org.xml.sax.ErrorHandler.error
未能转换 org.xml.sax.ErrorHandler.fatalError
未能转换 org.xml.sax.ErrorHandler.warning
未能转换 org.xml.sax.ext.DeclHandler
未能转换 org.xml.sax.ext.LexicalHandler
未能转换 org.xml.sax.HandlerBase
未能转换 org.xml.sax.helpers.AttributeListImpl.addAttribute
未能转换 org.xml.sax.helpers.AttributeListImpl.AttributeListImpl
未能转换 org.xml.sax.helpers.AttributeListImpl.AttributeListImpl(AttributeList)
未能转换 org.xml.sax.helpers.AttributeListImpl.clear
未能转换 org.xml.sax.helpers.AttributeListImpl.getLength
未能转换 org.xml.sax.helpers.AttributeListImpl.getName
未能转换 org.xml.sax.helpers.AttributeListImpl.getType
未能转换 org.xml.sax.helpers.AttributeListImpl.getValue
未能转换 org.xml.sax.helpers.AttributeListImpl.removeAttribute
未能转换 org.xml.sax.helpers.AttributeListImpl.setAttributeList
未能转换 org.xml.sax.helpers.AttributesImpl.addAttribute
未能转换 org.xml.sax.helpers.AttributesImpl.AttributesImpl
未能转换 org.xml.sax.helpers.AttributesImpl.AttributesImpl(Attributes)
未能转换 org.xml.sax.helpers.AttributesImpl.clear
未能转换 org.xml.sax.helpers.AttributesImpl.getIndex(String)
未能转换 org.xml.sax.helpers.AttributesImpl.getIndex(String, String)
未能转换 org.xml.sax.helpers.AttributesImpl.getLength

未能转换 org.xml.sax.helpers.AttributesImpl.getLocalName
未能转换 org.xml.sax.helpers.AttributesImpl.getQName
未能转换 org.xml.sax.helpers.AttributesImpl.getType
未能转换 org.xml.sax.helpers.AttributesImpl.getURI
未能转换 org.xml.sax.helpers.AttributesImpl.getValue
未能转换 org.xml.sax.helpers.AttributesImpl.removeAttribute
未能转换 org.xml.sax.helpers.AttributesImpl.setAttribute
未能转换 org.xml.sax.helpers.AttributesImpl.setAttributes
未能转换 org.xml.sax.helpers.AttributesImpl.setLocalName
未能转换 org.xml.sax.helpers.AttributesImpl.setQName
未能转换 org.xml.sax.helpers.AttributesImpl.setType
未能转换 org.xml.sax.helpers.AttributesImpl.setURI
未能转换 org.xml.sax.helpers.AttributesImpl.setValue
未能转换 org.xml.sax.helpers.NamespaceSupport.getDeclaredPrefixes
未能转换 org.xml.sax.helpers.NamespaceSupport.getPrefixes
未能转换 org.xml.sax.helpers.NamespaceSupport.getPrefixes(String)
未能转换 org.xml.sax.helpers.NamespaceSupport.processName
未能转换 org.xml.sax.helpers.NamespaceSupport.reset
未能转换 org.xml.sax.helpers.NamespaceSupport.XMLNS
未能转换 org.xml.sax.helpers.ParserAdapter
未能转换 org.xml.sax.helpers.ParserAdapter.getDTDHandler
未能转换 org.xml.sax.helpers.ParserAdapter.getEntityResolver
未能转换 org.xml.sax.helpers.ParserAdapter.getFeature
未能转换 org.xml.sax.helpers.ParserAdapter.getProperty
未能转换 org.xml.sax.helpers.ParserAdapter.setDTDHandler
未能转换 org.xml.sax.helpers.ParserAdapter.setErrorHandler
未能转换 org.xml.sax.helpers.ParserAdapter.setFeature
未能转换 org.xml.sax.helpers.ParserAdapter.setProperty
未能转换 org.xml.sax.helpers.XMLFilterImpl
未能转换 org.xml.sax.helpers.XMLReaderAdapter
未能转换 org.xml.sax.helpers.XMLReaderAdapter.setDTDHandler
未能转换 org.xml.sax.helpers.XMLReaderAdapter.setErrorHandler
未能转换 org.xml.sax.helpers.XMLReaderAdapter.setLocale
未能转换 org.xml.sax.helpers.XMLReaderFactory
未能转换 org.xml.sax.InputSource.getEncoding
未能转换 org.xml.sax.InputSource.getPublicId
未能转换 org.xml.sax.InputSource.setEncoding
未能转换 org.xml.sax.InputSource.setPublicId
未能转换 org.xml.sax.Parser.parse
未能转换 org.xml.sax.Parser.setDocumentHandler

未能转换 org.xml.sax.Parser.setDTDHandler
未能转换 org.xml.sax.Parser.setEntityResolver
未能转换 org.xml.sax.Parser.setErrorHandler
未能转换 org.xml.sax.Parser.setLocale
未能转换 org.xml.sax.SAXException
未能转换 org.xml.sax.SAXNotRecognizedException
未能转换 org.xml.sax.SAXNotSupportedException
未能转换 org.xml.sax.SAXParseException
未能转换 org.xml.sax.SAXParseException(SAXParseException, Locator)
未能转换 org.xml.sax.SAXParseException(SAXParseException, String, Locator, Exception)
未能转换 org.xml.sax.SAXParseException(SAXParseException, String, String, int, int)
未能转换 org.xml.sax.XMLFilter.getParent
未能转换 org.xml.sax.XMLFilter.setParent
未能转换 org.xml.sax.XMLReader.getContentHandler
未能转换 org.xml.sax.XMLReader.getDTDHandler
未能转换 org.xml.sax.XMLReader.getEntityResolver
未能转换 org.xml.sax.XMLReader.getErrorHandler
未能转换 org.xml.sax.XMLReader.getFeature
未能转换 org.xml.sax.XMLReader.getProperty
未能转换 org.xml.sax.XMLReader.parse
未能转换 org.xml.sax.XMLReader.setContentHandler
未能转换 org.xml.sax.XMLReader.setDTDHandler
未能转换 org.xml.sax.XMLReader.setEntityResolver
未能转换 org.xml.sax.XMLReader.setErrorHandler
未能转换 org.xml.sax.XMLReader.setFeature
未能转换 org.xml.sax.XMLReader.setProperty

C#(针对 C++ 开发人员)

下表包含 C# 和本机 C++(它不使用 /clr)之间的重要比较信息。如果您是一位 C++ 程序员，此表将为您提供这两种语言的最重要差异。

功能	参考主题
继承:在 C++ 中，类和结构实际上是相同的，而在 C# 中，它们很不一样。C# 类可以实现任意数量的接口，但只能从一个基类继承。而且，C# 结构不支持继承，也不支持显式默认构造函数(默认情况下提供一个)。	类 接口 struct(C# 参考)
数组:在 C++ 中，数组只是一个指针。在 C# 中，数组是包含方法和属性的对象。例如，可通过 Length 属性查询数组的大小。C# 数组还使用索引器(验证用于访问数组的各个索引)。声明 C# 数组的语法不同于声明 C++ 数组的语法:在 C# 中，“[]”标记出现在数组类型之后，而非变量之后。	数组(C# 编程指南) 索引器(C# 编程指南)
布尔值:在 C++ 中， bool 类型实质上是一个整数。在 C# 中，不存在 bool 类型与其他类型之间的相互转换。	bool
long 类型: long 类型在 C# 中为 64 位，而在 C++ 中为 32 位。	long
传递参数:在 C++ 中，除非显式通过指针或引用传递，否则所有变量都通过值传递。在 C# 中，除非显式通过具有 ref 或 out 参数修饰符的引用传递，否则类通过引用传递，而结构通过值传递。	结构 类 ref(C# 参考) out(C# 参考)
switch 语句:与 C++ 中的 switch 语句不同，C# 不支持从一个 case 标签贯穿到另一个 case 标签。	switch
委托:C# 委托大致类似于 C++ 中的函数指针，是类型安全和可靠的。	委托
基类方法:C# 支持用于调用派生类中重写基类成员的 base 关键字。而且，在 C# 中，使用 override 关键字重写虚拟或抽象方法是显式的。	base 请参见 override 的示例
方法隐藏:C++ 通过继承支持方法的隐式“隐藏”。在 C# 中，必须使用 new 修饰符来显式隐藏继承的成员。	new
预处理器指令用于条件编译。C# 中不使用头文件。	C# 预处理器指令
异常处理:无论是否引发异常，C# 都提供 finally 关键字以提供应执行的代码。	try-finally try-catch-finally

C# 运算符:C# 支持其他运算符, 如 is 和 typeof 。它还引入了某些逻辑运算符的不同功能。	& 运算符 运算符 ^ 运算符 为 typeof
extern 关键字:在 C++ 中, extern 用于导入类型。在 C# 中, extern 用于为使用同一程序集的不同版本创建别名。	extern
static 关键字:在 C++ 中, static 既可用于声明类级实体, 也可用于声明特定于某模块的类型。在 C# 中, static 仅用于声明类级实体。	static
C# 中的 Main 方法和 C++ 中的 main 函数的声明方式不同。在 C# 中, 它是大写的, 并且始终是 static 的。此外, 在 C# 中, 对处理命令行参数的支持要可靠得多。	Main() 和命令行参数 (C# 编程指南)
在 C# 中, 只有在 unsafe 模式下才允许使用指针。	unsafe
在 C# 中以不同的方式执行重载运算符。	C# 运算符
字符串:在 C++ 中, 字符串只是字符的数组。在 C# 中, 字符串是支持可靠搜索方法的对象。	字符串 String
foreach 关键字使您可以循环访问数组和集合。	foreach, in
全局:在 C# 中, 不支持全局方法和全局变量。方法和变量必须包含在 class 或 struct 之内。	C# 程序的常规结构
导入类型:在 C++ 中, 多个模块公用的类型放置在头文件中。在 C# 中, 可通过元数据获取此信息。	using 元数据概述
C# 中的局部变量在初始化前不能使用。	方法 (C# 编程指南)
内存管理:C++ 语言不提供垃圾回收功能;在进程终止前, 未显式释放的内存仍保持已分配的状态。C# 语言提供垃圾回收功能。	垃圾回收
析构函数:C# 具有用于确定地释放非托管资源的不同语法。	析构函数 using 语句 (C# 参考)
构造函数:与 C++ 类似, 如果在 C# 中不提供类构造函数, 则为您自动生成一个默认构造函数。该默认构造函数将所有字段初始化为它们的默认值。	实例构造函数 默认值表
C# 不支持位域。	C++ 位字段
C# 的输入/输出服务和格式设置依赖于 .NET Framework 的运行时库。	C# 语言教程 格式化数值结果表
在 C# 中, 方法参数不能有默认值。如果要获得同样的效果, 请使用方法重载。	编译器错误 CS0241
在 C# 中, 为类型参数化提供泛型类型和泛型方法的方式与 C++ 模板类似, 尽管存在显著的差异。	C# 中的泛型

as 关键字与标准强制转换类似，不同之处在于：如果转换失败，则返回值为空，而不是引发 [as \(C# 参考\)](#) 异常。这与使用 C++ 中的 **static_cast**（与 **dynamic_cast** 不同，它不执行运行时检查，因此失败时不引发异常）相似。

有关 C# 和其他编程语言中的关键字之间的比较的更多信息，请参见[对等语言](#)。有关 C# 应用程序的常规结构的信息，请参见[C# 程序的通用结构 \(C# 编程指南\)](#)。

请参见

概念

[C# 编程指南](#)

[项目中的项管理](#)

[使用解决方案资源管理器](#)

用 Visual C# 编写应用程序

C# 是一种类型安全的面向对象的语言，它简单易用，但功能强大，使程序员能够构建各种应用程序。通过结合 .NET Framework，可使用 Visual C# 创建 Windows 应用程序、Web 服务、数据库工具、组件、控件以及其他更多内容。

本节内容包含有关各种 Microsoft 平台技术的信息，您可以基于这些平台构建 C# 应用程序。

本节内容

[使用 .NET Framework 类库 \(Visual C#\)](#)

介绍如何在您的 Visual C# 项目中使用 .NET Framework 类库中的类型。

[创建 ASP.NET Web 应用程序 \(Visual C#\)](#)

介绍如何通过 Visual Web Developer 中的 C# 代码编辑器，使用 C# 语言创建具有代码隐藏页的 Web 应用程序。

[创建 Windows 窗体应用程序 \(Visual C#\)](#)

描述使用 Windows 窗体创建 Windows 应用程序。

[创建控制台应用程序 \(Visual C#\)](#)

解释如何创建无需图形用户界面的应用程序。

[访问和显示数据 \(Visual C#\)](#)

描述与数据库的交互。

[创建移动应用程序和嵌入式应用程序 \(Visual C#\)](#)

介绍如何针对智能设备、嵌入设备以及瘦移动客户端创建应用程序。

[创建和访问 Web 服务 \(Visual C#\)](#)

介绍如何与 XML Web services 进行交互。

[创建组件 \(Visual C#\)](#)

介绍如何为 .NET Framework 创建用户控件和其他组件。

[在 Office 平台上进行开发 \(Visual C#\)](#)

介绍如何使用 Visual Studio Tools for Office 创建智能文档。

[用于企业开发 \(Visual C#\)](#)

介绍如何针对 SQL Server、Microsoft Exchange Server 以及其他产品开发应用程序。

[Tablet PC 编程 \(Visual C#\)](#)

介绍如何为 Tablet PC 开发基于墨迹的应用程序。

[音频、视频、游戏和图形 \(Visual C#\)](#)

介绍如何在托管代码中使用 Windows Media 和 DirectX。

[创建初学者工具包 \(Visual C#\)](#)

介绍如何创建初学者工具包，通过示例代码帮助他人迅速掌握新的知识并运用于工作之中。

请参见

其他资源

[Visual C#](#)

[C# 参考](#)

[Visual C# 入门](#)

[使用 Visual C# IDE](#)

使用 .NET Framework 类库 (Visual C#)

多数 Visual C# 开发项目都将 .NET Framework 类库广泛用于各个方面，从文件系统访问和字符串操作到 Windows 窗体和 ASP.NET 用户界面控件。

该类库被组织到多个命名空间中，每个命名空间包含一组相关的类和结构。例如，[System.Drawing](#) 命名空间包含多种类型，这些类型表示字体、画笔、线条、形状、颜色等。

using 指令和引用

必须将给定命名空间的 [using 指令](#) 添加到 C# 源文件中，才能在 C# 程序中使用该命名空间中的类。在某些情况下，还必须添加对包含该命名空间的 DLL 的引用；Visual C# 自动添加对最常用的类库 DLL 的引用。您可以在“引用”节点下的“解决方案资源管理器”中查看添加了哪些引用。有关更多信息，请参见[创建项目 \(Visual C#\)](#)。

添加命名空间的 [using](#) 指令后，可以创建该命名空间中类型的实例、调用方法、以及事件响应，就像它们已在各自的源代码中进行了声明一样。在 Visual C# 代码编辑器中，还可以将插入点置于类型或成员名称上，然后按 F1 查看“帮助”文档。也可以使用“对象浏览器”工具和“用作源代码的元数据”功能查看有关 .NET Framework 类和结构的类型信息。有关更多信息，请参见[建模和分析代码 \(Visual C#\)](#)。

有关更多信息

- 有关 .NET Framework 类库的更多信息，请参见 [.NET Framework 类库概述](#) 和 [.NET Framework 编程](#)。
- 有关 .NET Framework 架构的更多信息，请参见 [.NET Framework 概述](#)。
- 在 Internet 上，[.NET Framework Developer Center](#) 有很多关于类库的文章和代码示例。
- 有关如何使用类库执行特定任务的信息，请参见[如何实现 - C#](#)，或在 Visual C# 中单击“帮助”菜单上的“如何实现”。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

[使用 Visual C# IDE](#)

创建 ASP.NET Web 应用程序 (Visual C#)

ASP.NET 提供了一个统一的 Web 开发模型，其中包括生成企业级 Web 应用程序所必需的各种服务。ASP.NET 是 .NET Framework 的组成部分，使您能够充分利用公共语言运行库 (CLR) 的功能，如类型安全、继承、语言互操作性和版本控制。

当您使用 Visual Studio 创建 ASP.NET 网站时，实际上就是在使用集成开发环境 (IDE) 的一部分，也称为 Visual Web Developer。Visual Web Developer 与 Visual C# 不同；它拥有自己的设计器，用于在网页上创建用户界面，还拥有进行 Web 开发和网站管理的其他工具。但当您在 C# 中创建 Web 控件的代码隐藏页时，就是在使用 C# 代码编辑器，并且可以像在 Visual C# 中那样，在 Visual Web Developer 中使用编辑器的所有功能。

有关创建 ASP.NET 网站和网页的更多信息，请参见 [Visual Web Developer](#)。

MSDN Online 上的 [ASP.NET Developer Center\(ASP.NET 开发人员中心\)](#) 中有很多并未包括在产品文档中的有用的文章。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

创建 Windows 窗体应用程序 (Visual C#)

Windows 窗体技术可用于在 Visual C# 中创建运行于 .NET Framework 上的基于 Windows 的智能客户端应用程序。创建 Windows 应用程序项目时，您正在创建基于 Windows 窗体的应用程序。您将使用 [Windows 窗体设计器](#) 创建自己的用户界面，还将访问其他设计功能和运行时功能，其中包括：

- [ClickOnce 部署](#)。
- 利用 [DataGridView 控件](#) 实现的强大数据库支持。
- 具有 Microsoft® Windows® XP、Microsoft Office 或 Microsoft Internet Explorer 的外观和行为的工具栏和其他用户界面元素。

有关更多信息，请参见[设计用户界面 \(Visual C#\)](#) 和 [Windows 窗体](#)。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

创建控制台应用程序 (Visual C#)

可以使用 C# 创建在命令行控制台接收输入并显示输出的应用程序。因为这些应用程序的用户界面非常简单，所以对于学习 C# 开发非常理想。控制台应用程序对于极少需要或不需要用户交互的实用工具程序也非常有用。

若要用 Visual C# 创建控制台应用程序，请单击“文件”菜单上的“新建”，然后选择“项目”。单击“C# 控制台应用程序”项目模板，输入您要使用的文件名，然后单击“确定”。

使用 System.Console 类

使用 [Console](#) 类可以单个字符或整行方式从控制台中读取字符并向其中写入字符。可以用多种不同方式设置输出格式。有关更多信息，请参见[格式化概述](#)。

可以使用与 `Main` 方法关联的可选字符串数组访问命令行参数。有关更多信息，请参见[命令行参数 \(C# 编程指南\)](#)。

示例

- [如何：创建控制台应用程序客户端](#)
- [“命令行参数”示例](#)

请参见

其他资源

[用 Visual C# 编写应用程序](#)

访问和显示数据 (Visual C#)

在 Visual C# 应用程序中，您将通常使用在 [System.Data](#) 和 .NET Framework 类库中相关的命名空间中公开的 [ADO.NET](#) 技术与数据库进行连接。有关使用数据的更多信息，请参见[访问数据 \(Visual Studio\)](#)。

在 Windows 窗体应用程序中，用于显示从数据库中检索到的数据的主要用户界面控件为 DataGridView。有关更多信息，请参见[DataGridView 控件 \(Windows 窗体\)](#)。通过所谓的数据绑定功能，可以极大地简化将数据源连接到用户界面控件(如文本框和列表框)的操作。将控件绑定到数据源中的字段后，对其中一项的更改会自动反映在另一项上。有关更多信息，请参见[Windows 窗体数据绑定](#)。

有关创建和管理数据库、编写存储过程的更多信息以及其他相关信息，请参见[SQL Server 项目](#)和[SQL Server 教程](#)。

以下链接包含使用 Visual Studio 访问数据的信息：

- [使用 Visual Studio 创建数据应用程序](#)
- [Visual Database Tools](#)

在 Internet 上，[Data Access and Storage Developer Center \(数据访问和存储开发人员中心\)](#)会不断更新，添加新文章和新示例。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

创建移动应用程序和嵌入式应用程序 (Visual C#)

对于开发运行在基于 Windows CE 的智能设备(如 Pocket PC 和 Smartphone)上的软件, Microsoft Visual Studio 2005 提供充分的集成化的支持。可以使用 Visual C# 编写在 .NET Compact Framework 上运行的托管应用程序。还可使用开发 PC 软件时使用的相同代码编辑器、设计器和调试器界面。只需选择您使用的语言可用的“智能设备”项目模板, 然后就可以开始编码。

Visual Studio 还为智能设备提供仿真程序, 允许在开发计算机上运行和调试代码;还提供工具, 简化将应用程序及其资源打包到 CAB 文件中以部署到最终用户设备的过程。有关更多信息, 请参见 [智能设备](#)。

也可以使用 [Visual Web Developer](#) 开发基于 ASP.NET 的移动 Web 应用程序。有关更多信息, 请参见[ASP.NET 移动网页介绍](#)。

若要了解有关 Windows Mobile 技术的最新信息, 请访问 [Mobile Developer Center \(Mobile 开发人员中心\)](#)。

[请参见](#)

[其他资源](#)

[用 Visual C# 编写应用程序](#)

创建和访问 Web 服务 (Visual C#)

在松耦合环境中, XML Web services 为应用程序提供使用围绕标准协议(例如 HTTP、XML、XSD、SOAP 和 WSDL)生成的预定消息交换进行通信的能力。因为协议和规范是公共的且并不特定于平台, 所以 XML Web services 允许应用程序进行通信, 不论它们是否驻留在同一台计算机上, 甚至驻留在不同的计算平台或设备上也可以进行通信。

您不需要深入了解每个规范, 就能生成或使用 Web 服务。.NET Framework 类和 Visual Studio 向导会帮助您使用熟悉的面向对象编程模型, 生成 XML Web services 或与之交互。

有关 XML Web services 的更多信息, 请参见 [XML Web 服务概述](#)。

有关使用 ASP.NET 创建和访问 XML Web services 的更多信息, 请参见[使用 ASP.NET 的 XML Web 服务](#)。

有关 Visual Studio 提供的帮助您轻松生成和使用 XML Web services 的工具的更多信息, 请参见
[XML Web services \(Visual Studio\)](#)。

有关创建和访问 XML Web Services 的更多信息, 请参见[在托管代码中使用的 XML Web services](#)。

若要获得其他文章和资源, 请访问 MSDN Online 上的 [Web Services Developer Center \(Web 服务开发人员中心\)](#)。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

创建组件 (Visual C#)

软件行业中的术语“组件”常用于指可重用的、以标准化方式向客户端公开一个或多个接口的对象。一个组件可作为一个类实现，也可作为一组类实现；主要要求是完善定义基本的公共接口。例如，在本机 Windows 编程上下文中，组件对象模型 (COM) 要求所有组件除了实现任何其他专用接口外，还需实现 [IUnknown](#) 接口。

在 .NET Framework 的上下文中，组件是实现 [IComponent](#) 接口的一个类或一组类，或者是直接或间接由实现此接口的类派生出的一个类。**IComponent** 接口的默认基类实现为 [Component](#)。

在 .NET Framework 编程中最常用的一些组件是添加到 Windows 窗体中的可视控件，如 [Button 控件 \(Windows 窗体\)](#)、[ComboBox 控件 \(Windows 窗体\)](#) 等。非可视组件包括 [Timer Control](#)、[SerialPort](#) 和 [ServiceController](#) 以及其他组件。

在 C# 中创建组件时，使用任何符合[公共语言规范](#)的其他语言编写的客户端都可使用该组件。

若要在 Visual C# 中创建自己的组件，可以使用[组件设计器](#)以组合 Windows 窗体的方式组合非可视组件类。有关更多信息，请参见[演练:在组件设计器中创建 Windows 服务应用程序](#)。

有关使用 Visual Studio 进行组件编程的更多信息，请参见[Visual Studio 中的组件](#)。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

在 Office 平台上进行开发 (Visual C#)

Microsoft Visual Studio 2005 Tools for the Microsoft Office System 允许您使用托管代码自定义 Microsoft Office 文档和 Microsoft Office Outlook。Visual Studio Tools for Office 为 Visual Studio 增添了新的功能，例如：在 Visual Studio 开发环境中将 Microsoft Office Word 和 Microsoft Office Excel 作为设计器进行承载，直接对数据和呈现对象进行编程的能力，以及在文档上和在“文档操作”任务窗格中使用 Windows 窗体控件的能力。

有关在 Office 平台上进行开发的详细信息，请参见以下主题：

[Visual Studio Tools for Office 中的新增功能](#)

[Excel 文档级自定义项编程入门](#)

[Word 文档级自定义项编程入门](#)

[应用程序级外接程序编程入门](#)

[Office 编程中的常见任务](#)

请参见

其他资源

[用 Visual C# 编写应用程序](#)

用于企业开发 (Visual C#)

可以在 C# 中编程，为企业开发在基于服务器的结构和服务器产品上运行的多种应用程序。MSDN Library 中的 [Servers and Enterprise Development\(服务器和企业开发\)](#) 章节包含了结构指南、模式和实践信息，以及使用 Microsoft 服务器产品的文档和技术文章，其中包括：

- Microsoft SQL Server
- Microsoft BizTalk Server
- Microsoft Commerce Server
- Microsoft Content Management Server
- Microsoft Exchange Server
- Microsoft Host Integration Server
- Microsoft Internet Security and Acceleration Server 2000
- Microsoft Business Solutions
- Microsoft MapPoint
- Microsoft Speech Server

请参见

其他资源

[用 Visual C# 编写应用程序](#)

Tablet PC 编程 (Visual C#)

Tablet PC 是使用 Microsoft® Windows® XP 的个人计算机，适合于支持墨水、手写笔和语音的应用程序。在 Tablet PC 中，软件和硬件的结合支持这些用户交互方法，并且向用户提供了丰富高效的交互式计算体验。

Tablet PC 平台包括 Windows XP 及其扩展，它们支持在 Tablet PC 上输入和输出手写和语音数据，以及与其他计算机相互交换这些数据。

有关更多信息，请参见 MSDN Online 上的 [Windows XP Tablet PC Edition](#) 主题以及访问 [Tablet 和 Mobile PC 开发人员中心](#)。

请参见

其他资源

[用 Visual C# 编写应用程序](#)

[Tablet 和 Mobile PC 开发人员中心](#)

音频、视频、游戏和图形 (Visual C#)

可以使用 Visual C# 创建基于 DirectX for Managed Code 和 Windows Media Technologies 的多媒体应用程序。

Managed DirectX

Microsoft® DirectX® 是多媒体应用程序编程接口 (API) 的高级套件，内置于 Microsoft Windows® 操作系统中。DirectX 允许软件开发人员访问专用的硬件功能，而无需编写特定于硬件的代码，从而为基于 Windows 的 PC 提供标准开发平台。此技术在 1995 年首次引入，随即成为在 Windows 平台上进行多媒体应用程序开发的公认标准。

简单地说，DirectX 是一项 Windows 技术，在 PC 上玩游戏或观看视频时，可以实现更高性能的图形（包括全彩色图形）、视频和 3-D 动画以及声音（包括环场声音）。

有关在 C# 应用程序中使用 DirectX 的更多信息，请参见位于 MSDN Online 上的 [DirectX 9.0 for Managed Code \(托管代码的 DirectX 9.0 编程\)](#) 以及访问 [Microsoft DirectX 开发人员中心](#)

Windows Media Player

Windows Media Player ActiveX 控件可以用于 C# 应用程序中，以添加音频和视频播放功能。Microsoft Windows Media Player 10 软件开发工具包 (SDK) 提供了自定义 Windows Media Player 以及使用 Windows Media Player ActiveX 控件的信息和工具。该 SDK 包括说明如何在 C# 应用程序中使用 Media Player ActiveX 控件的文档和代码示例。

有关更多信息，请参见位于 MSDN Online 上的 [Windows Media Player 10 SDK](#)。

Windows Media Encoder

Windows Media Encoder 9 系列 SDK 可以使用 C# 编程，以执行以下类型的任务：

- **广播实况内容。**新闻单位可以使用自动化 API 安排自动捕获和广播实况内容。例如，地方交通部门可以在多个事故地点以数据流方式传输路况的实况图像，警告驾驶员注意交通拥堵并建议他们选择其他的路线。
- **批处理内容。**必须处理大量大型文件的媒体生产单位可以创建批处理，使用自动化 API 连续反复捕获数据流并对其进行编码。企业可以使用自动化 API、通过首选的脚本撰写语言和 Windows 脚本宿主对其流媒体服务进行管理。Windows 脚本宿主是一种不依赖于语言的宿主，可用于在 Microsoft Windows® 95 或更高版本、Windows NT 或 Windows 2000 操作系统上运行任何脚本引擎。
- **创建自定义用户界面**Internet 服务提供商 (ISP) 可以生成使用自动化 API 的功能捕获、编码和广播媒体流的界面。或者，也可以使用自动化 API 中的预定义用户界面实现同一目的。
- **远程管理 Windows Media Encoder 应用程序。**可以使用自动化 API 从远程计算机运行、故障排除和管理 Windows Media Encoder 应用程序。

有关更多信息，请参见位于 MSDN online 上的 [Windows Media Encoder 9 Series SDK](#)。[Programming C#](#) 主题描述了使用 C# 时要包括哪些引用。

Windows Media Server

Microsoft® Windows Media® Services 9 系列软件开发工具包 (SDK) 是一种功能强大的基于自动化的应用程序编程接口 (API)，专为希望开发 Windows Media Services 9 系列应用程序的人员设计。可以在 C# 中使用此 SDK 以编程方式管理 Windows Media Server，向启用单播和多播的网络上的客户端发送数字媒体内容。有关更多信息，请参见 [Windows Media Services 9 Series SDK](#)。

请参见

其他资源

用 Visual C# 编写应用程序

创建初学者工具包 (Visual C#)

初学者工具包中包含了完整的应用程序代码，以及有关如何修改或扩展应用程序的文档。Visual C# 2005 提供了两个初学者工具包，可以在“新建项目”对话框中访问。也可以创建自己的初学者工具包，帮助用户在任何类型的应用程序上快速进入工作状态。创建初学者工具包与创建标准项目模板基本相同，唯一的差别在于，创建基于初学者工具包的项目时，初学者工具包包含的文档文件被设置为打开状态。

有关更多信息，请参见[初学者工具包介绍](#)。

[请参见](#)

[其他资源](#)

[用 Visual C# 编写应用程序](#)

C# 编程指南

本节提供有关关键的 C# 语言功能和 C# 可通过 .NET Framework 访问的功能的详细信息。

语言部分

[在 C# 程序内部](#)

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

[数据类型 \(C# 编程指南\)](#)

[数组 \(C# 编程指南\)](#)

[字符串 \(C# 编程指南\)](#)

[语句、表达式和运算符 \(C# 编程指南\)](#)

[对象、类和结构 \(C# 编程指南\)](#)

[属性 \(C# 编程指南\)](#)

[索引器 \(C# 编程指南\)](#)

[委托 \(C# 编程指南\)](#)

[事件 \(C# 编程指南\)](#)

[泛型 \(C# 编程指南\)](#)

[迭代器 \(C# 编程指南\)](#)

[命名空间 \(C# 编程指南\)](#)

[可空类型 \(C# 编程指南\)](#)

[不安全代码和指针 \(C# 编程指南\)](#)

[XML 文档注释 \(C# 编程指南\)](#)

平台部分

[应用程序域 \(C# 编程指南\)](#)

[程序集和全局程序集缓存 \(C# 编程指南\)](#)

[属性 \(C# 编程指南\)](#)

[集合类 \(C# 编程指南\)](#)

[异常和异常处理 \(C# 编程指南\)](#)

[互操作性 \(C# 编程指南\)](#)

[线程处理 \(C# 编程指南\)](#)

[性能 \(C# 编程指南\)](#)

[反射 \(C# 编程指南\)](#)

[C# DLL \(C# 编程指南\)](#)

[安全性 \(C# 编程指南\)](#)

[请参见](#)

[其他资源](#)

[C# 参考](#)

[Visual C#](#)

在 C# 程序内部

本节讨论 C# 程序的一般结构，还包括标准的“Hello, World!”示例。

本节内容

- [Hello World -- 您的第一个程序 \(C# 编程指南\)](#)
- [C# 程序的通用结构 \(C# 编程指南\)](#)

相关章节

- [Visual C# 入门](#)
- [迁移到 Visual C#](#)
- [C# 编程指南](#)
- [C# 参考](#)
- [Visual C# 示例](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1 简介](#)

请参见

概念

[C# 编程指南](#)

Hello World -- 您的第一个程序(C# 编程指南)

以下控制台程序是传统“Hello World!”程序的 C# 版，该程序显示字符串 Hello World!。

C#

```
using System;
// A "Hello World!" program in C#
namespace HelloWorld
{
    class Hello
    {
        static void Main()
        {
            System.Console.WriteLine("Hello World!");
        }
    }
}
```

现在依次介绍此程序的重要组成部分。

注释

第一行包含注释语句：

C#

```
// A "Hello World!" program in C#
```

// 字符将这行的其余内容转换为注释内容。还可通过将文本块置于 /* 和 */ 字符之间来注释文本块，如下所示：

C#

```
/* A "Hello World!" program in C#.
This program displays the string "Hello World!" on the screen. */
```

Main 方法

C# 程序必须包含一个 `Main` 方法，程序控制在该方法中开始和结束。在 `Main` 方法中创建对象和执行其他方法。

`Main` 方法是驻留在类或结构内的静态方法。在前面的“Hello World!”示例中，它驻留在名为 `Hello` 的类中。用下列方式之一声明 `Main` 方法：

- 该方式返回 `void`：

C#

```
static void Main()
{
    //...
}
```

- 它还可以返回 `int`：

C#

```
static int Main()
{
    //...
    return 0;
```

```
}
```

- 由于有这两个返回类型，它可以带有参数：

```
C#
```

```
static void Main(string[] args)
{
    //...
}
```

- 或 -

```
C#
```

```
static int Main(string[] args)
{
    //...
    return 0;
}
```

Main 方法的参数是 string 数组，该数组表示用于激活程序的命令行参数。请注意，不像 C++，该数组不包含可执行 (exe) 文件名。

有关使用命令行参数的更多信息，请参见 [Main\(\) 和命令行参数 \(C# 编程指南\)](#) 和 [如何：创建和使用 C# DLL \(C# 编程指南\)](#) 中的示例。

输入和输出

C# 程序通常使用 .NET Framework 的运行时库提供的输入/输出服务。语句 `System.Console.WriteLine("Hello World!");` 使用了 [WriteLine](#) 方法，该方法是运行时库中的 `Console` 类的输出方法之一。它显示了标准输出流使用的字符串参数，输出流后面跟一个新行。其他 `Console` 方法用于不同的输入和输出操作。如果程序开始处包含 `using System;` 指令，则无需完全限定 `System` 类和方法即可直接使用它们。例如，您可以改为调用 `Console.WriteLine`，而不必指定 `System.Console.WriteLine`：

```
C#
```

```
using System;
```

```
C#
```

```
Console.WriteLine("Hello World!");
```

有关输入/输出方法的更多信息，请参见 [System.IO](#)。

编译和执行

可以通过在 Visual Studio IDE 中创建项目或使用命令行来编译“Hello World!”程序。使用 Visual Studio 命令提示窗口或调用 `vsvars32.bat` 将 Visual C# 工具集放置在命令提示符下的路径中。

从命令行编译程序：

- 使用文本编辑器创建源文件，并将其存储为名如 `Hello.cs` 的文件。C# 源代码文件使用的扩展名是 `.cs`。
- 若要激活编译器，请输入命令：

```
csc Hello.cs
```

如果程序没有包含任何编译错误，则将创建一个 `Hello.exe` 文件。

- 若要运行程序，请输入命令：

Hello

有关 C# 编译器及其选项的更多信息, 请参见 [C# 编译器选项](#)。

请参见

参考

[在 C# 程序内部](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

C# 参考

C# 程序的通用结构 (C# 编程指南)

C# 程序可由一个或多个文件组成。每个文件都可以包含零个或零个以上的命名空间。一个命名空间除了可包含其他命名空间外，还可包含类、结构、接口、枚举、委托等类型。以下是 C# 程序的主干，它包含所有这些元素。

C#

```
// A skeleton of a C# program
using System;
namespace YourNamespace
{
    class YourClass
    {
    }

    struct YourStruct
    {
    }

    interface IYourInterface
    {
    }

    delegate int YourDelegate();

    enum YourEnum
    {
    }

    namespace YourNestedNamespace
    {
        struct YourStruct
        {
        }
    }

    class YourMainClass
    {
        static void Main(string[] args)
        {
            //Your program starts here...
        }
    }
}
```

[相关章节](#)

[更多信息：](#)

- [类\(C# 编程指南\)](#)
- [结构\(C# 编程指南\)](#)
- [命名空间\(C# 编程指南\)](#)
- [接口\(C# 编程指南\)](#)
- [委托\(C# 编程指南\)](#)

[C# 语言规范](#)

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1.2 程序结构](#)
- [9.1 编译单元\(命名空间\)](#)

[请参见](#)

[参考](#)

[在 C# 程序内部](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[Visual C# 示例](#)

Main() 和命令行参数 (C# 编程指南)

Main 方法是程序的入口点，您将在那里创建对象和调用其他方法。一个 C# 程序中只能有一个入口点。

C#

```
class TestClass
{
    static void Main(string[] args)
    {
        // Display the number of command line arguments:
        System.Console.WriteLine(args.Length);
    }
}
```

概述

- **Main** 方法是程序的入口点，程序控制在该方法中开始和结束。
- 该方法在类或结构的内部声明。它必须为静态方法，而不应为公共方法。(在上面的示例中，它接受默认访问级别 `private`。)
- 它可以具有 `void` 或 `int` 返回类型。
- 声明 **Main** 方法时既可以使用参数，也可以不使用参数。
- 参数可以作为从零开始索引的命令行参数来读取。
- 与 C 和 C++ 不同，程序的名称不会被当作第一个命令行参数。

本节内容

- [命令行参数 \(C# 编程指南\)](#)
- [如何：显示命令行参数 \(C# 编程指南\)](#)
- [如何：使用 foreach 访问命令行参数 \(C# 编程指南\)](#)
- [Main\(\) 返回值 \(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.1 Hello World

请参见

参考

[在 C# 程序内部](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

命令行参数 (C# 编程指南)

`Main` 方法可以使用参数，在这种情况下它采用下列形式之一：

C#

```
static int Main(string[] args)
```

C#

```
static void Main(string[] args)
```

`Main` 方法的参数是表示命令行参数的 `String` 数组。通常通过测试 `Length` 属性来检查参数是否存在，例如：

C#

```
if (args.Length == 0)
{
    System.Console.WriteLine("Please enter a numeric argument.");
    return 1;
}
```

还可以使用 `Convert` 类或 `Parse` 方法将字符串参数转换为数值类型。例如，下面的语句使用 `Int64` 类上的 `Parse` 方法将字符串转换为 `long` 型数字：

```
long num = Int64.Parse(args[0]);
```

也可以使用别名为 `Int64` 的 C# 类型 `long`：

```
long num = long.Parse(args[0]);
```

还可以使用 `Convert` 类的方法 `ToInt64` 完成同样的工作：

```
long num = Convert.ToInt64(s);
```

有关更多信息，请参见 [Parse](#) 和 [Convert](#)。

示例

在此示例中，程序在运行时采用一个参数，将该参数转换为整数，并计算该数的阶乘。如果没有提供参数，则程序发出一条消息来解释程序的正确用法。

C#

```
// arguments: 3
```

C#

```
public class Functions
{
    public static long Factorial(int n)
    {
        if (n < 0) { return -1; }      //error result - undefined
        if (n > 256) { return -2; }   //error result - input is too big
        if (n == 0) { return 1; }

        // Calculate the factorial iteratively rather than recursively:
```

```
        long tempResult = 1;
        for (int i = 1; i <= n; i++)
        {
            tempResult *= i;
        }
        return tempResult;
    }
}
```

C#

```
class MainClass
{
    static int Main(string[] args)
    {
        // Test if input arguments were supplied:
        if (args.Length == 0)
        {
            System.Console.WriteLine("Please enter a numeric argument.");
            System.Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }

        try
        {
            // Convert the input arguments to numbers:
            int num = int.Parse(args[0]);

            System.Console.WriteLine("The Factorial of {0} is {1}.", num, Functions.Factorial(num));
            return 0;
        }
        catch (System.FormatException)
        {
            System.Console.WriteLine("Please enter a numeric argument.");
            System.Console.WriteLine("Usage: Factorial <num>");
            return 1;
        }
    }
}
```

输出

The Factorial of 3 is 6.

注释

下面是该程序的两个运行示例(假定程序名为 Factorial.exe)。

运行示例 #1:

输入下面的命令行:

Factorial 10

您将获得下面的结果:

The Factorial of 10 is 3628800.

运行示例 #2:

输入下面的命令行:

Factorial

您将获得下面的结果:

Please enter a numeric argument.

Usage: Factorial <num>

有关命令行参数的更多用法示例，请参见[如何：创建和使用 C# DLL \(C# 编程指南\)](#)中的示例。

请参见

任务

[如何：显示命令行参数 \(C# 编程指南\)](#)

[如何：使用 foreach 访问命令行参数 \(C# 编程指南\)](#)

参考

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

[Main\(\) 返回值 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

如何: 显示命令行参数 (C# 编程指南)

可以通过 `Main` 的可选参数来访问通过命令行提供给可执行文件的参数。参数以字符串数组的形式提供。数组的每个元素都包含一个参数。参数之间的空白被移除。例如，下面是对一个假想的可执行文件的命令行调用：

命令行输入	传递给 <code>Main</code> 的字符串数组
<code>executable.exe a b c</code>	"a" "b" "c"
<code>executable.exe one two</code>	"one" "two"
<code>executable.exe "one two" three</code>	"one two" "three"

示例

本示例显示了传递给命令行应用程序的命令行参数。显示的输出对应于上表中的第一项。

C#

```
class CommandLine
{
    static void Main(string[] args)
    {
        // The Length property provides the number of array elements
        System.Console.WriteLine("parameter count = {0}", args.Length);

        for (int i = 0; i < args.Length; i++)
        {
            System.Console.WriteLine("Arg[{0}] = [{1}]", i, args[i]);
        }
    }
}
```

输出

```
parameter count = 3
Arg[0] = [a]
Arg[1] = [b]
Arg[2] = [c]
```

请参见

任务

[如何: 使用 foreach 访问命令行参数 \(C# 编程指南\)](#)

参考

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

[Main\(\) 返回值 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

如何：使用 foreach 访问命令行参数 (C# 编程指南)

循环访问数组的另一种方法是使用 `foreach` 语句，如下面的示例所示。`foreach` 语句可以用于循环访问数组、.NET Framework 集合类或任何实现 `IEnumerable` 接口的类或结构。

示例

下面的示例演示如何使用 `foreach` 输出命令行参数。

C#

```
// arguments: John Paul Mary
```

C#

```
class CommandLine2
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Number of command line parameters = {0}", args.Length);

        foreach (string s in args)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

输出

```
Number of command line parameters = 3
John
Paul
Mary
```

请参见

任务

[如何：显示命令行参数 \(C# 编程指南\)](#)

参考

[foreach, in \(C# 参考\)](#)

[Main\(\) 和命令行参数 \(C# 编程指南\)](#)

[Main\(\) 返回值 \(C# 编程指南\)](#)

[Array](#)

[System.Collections](#)

概念

[C# 编程指南](#)

Main() 返回值 (C# 编程指南)

Main 方法可以是 **void** 类型:

C#

```
static void Main()
{
    //...
}
```

它还可以返回 **int**:

C#

```
static int Main()
{
    //...
    return 0;
}
```

如果不需要使用 Main 的返回值，则返回 **void** 可以使代码变得略微简单。但是，返回整数可使程序将状态信息与调用该可执行文件的其他程序或脚本相关。下面的示例演示使用 Main 的返回值。

示例

在此示例中，使用了一个批处理文件来执行程序并测试 Main 函数的返回值。在 Windows 中执行程序时，Main 函数返回的任何值都将存储在名为 **ERRORLEVEL** 的环境变量中。通过检查 **ERRORLEVEL** 变量，批处理文件可以确定执行的结果。通常，返回值为零指示执行成功。下面是一个非常简单的程序，其 Main 函数返回零。

C#

```
class MainReturnValueTest
{
    static int Main()
    {
        //...
        return 0;
    }
}
```

由于此示例使用了批处理文件，因此最好从命令行编译这段代码，如[如何：设置环境变量](#)中所示。

然后，使用批处理文件调用前面的代码示例所生成的可执行文件。由于代码返回零，因此批处理文件将报告成功。但如果前面的代码更改为返回非零值，然后重新编译，则批处理文件的后续执行将指示失败。

```
rem test.bat
@echo off
MainReturnValueTest
@if "%ERRORLEVEL%" == "0" goto good

:fail
    echo Execution Failed
    echo return value = %ERRORLEVEL%
    goto end

:good
    echo Execution Succeeded
    echo return value = %ERRORLEVEL%
    goto end

:end
```

示例输出

```
Execution Succeeded
```

```
return value = 0
```

请参见

任务

[如何:显示命令行参数\(C# 编程指南\)](#)

[如何:使用 foreach 访问命令行参数\(C# 编程指南\)](#)

参考

[Main\(\) 和命令行参数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

数据类型(C# 编程指南)

C# 是强类型语言;因此每个变量和对象都必须具有声明类型。

数据类型概述

数据类型可描述为:

- 内置数据类型, 如 `int` 或 `char`,
- 用户定义数据类型, 如 `class` 或 `interface`。
- 数据类型还可以定义为:
 - 值类型(C# 参考)(用于存储值),
 - 引用类型(C# 参考)(用于存储对实际数据的引用)。

相关章节

更多信息:

- [强制转换\(C# 编程指南\)](#)
- [装箱和取消装箱\(C# 编程指南\)](#)
- [类型\(C# 参考\)](#)
- [对象、类和结构\(C# 编程指南\)](#)

C# 语言规范

有关类型的更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 1.3 类型和变量
- 3.8 命名空间和类型名称
- 4.1 值类型
- 4.2 引用类型
- 4.3 装箱和取消装箱

请参见

参考

[不同语言的数据类型比较](#)

[整型表\(C# 参考\)](#)

概念

[C# 编程指南](#)

[XML 数据类型的转换](#)

其他资源

[C# 参考](#)

强制转换(C# 编程指南)

数据类型之间的转换可以使用强制转换显式进行，但在某些情况下，也允许隐式转换。例如：

C#

```
static void TestCasting()
{
    int i = 10;
    float f = 0;
    f = i; // An implicit conversion, no data will be lost.
    f = 0.5F;
    i = (int)f; // An explicit conversion. Information will be lost.
}
```

显式强制转换调用转换运算符，从一种类型转换为另一种类型。如果未定义相应的转换运算符，则强制转换会失败。可以编写自定义转换运算符，在用户定义类型之间进行转换。有关定义转换运算符的更多信息，请参见 [explicit\(C# 参考\)](#) 和 [implicit\(C# 参考\)](#)。

示例

下面的程序将 `double` 强制转换为 `int`。如不强制转换则该程序不会进行编译。

C#

```
class Test
{
    static void Main()
    {
        double x = 1234.7;
        int a;
        a = (int)x; // cast double to int
        System.Console.WriteLine(a);
    }
}
```

输出

1234

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.6.6 强制转换表达式
- 6.1 隐式转换
- 6.2 显式转换

请参见

参考

[数据类型\(C# 编程指南\)](#)[\(\) 运算符\(C# 参考\)](#)[explicit\(C# 参考\)](#)[implicit\(C# 参考\)](#)

概念

[C# 编程指南](#)[转换运算符\(C# 编程指南\)](#)[通用类型转换](#)[显式转换](#)[导出类型转换](#)

装箱和取消装箱(C# 编程指南)

装箱和取消装箱使值类型能够被视为对象。对值类型装箱将把该值类型打包到 `Object` 引用类型的一个实例中。这使得值类型可以存储于垃圾回收堆中。取消装箱将从对象中提取值类型。在此示例中，整型变量 `i` 被“装箱”并赋值给对象 `o`。

C#

```
int i = 123;
object o = (object)i; // boxing
```

然后，可以对对象 `o` 取消装箱并将其赋值给整型变量 `i`：

C#

```
o = 123;
i = (int)o; // unboxing
```

性能

相对于简单的赋值而言，装箱和取消装箱过程需要进行大量的计算。对值类型进行装箱时，必须分配并构造一个全新的对象。次之，取消装箱所需的强制转换也需要进行大量的计算。有关更多信息，请参见[性能](#)。

相关章节

更多信息：

- [装箱转换](#)
- [取消装箱转换](#)
- [引用类型](#)
- [值类型](#)

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 4.3 装箱和取消装箱

请参见

概念

[C# 编程指南](#)

[已装箱的值类型](#)

装箱转换(C# 编程指南)

装箱用于在垃圾回收堆中存储值类型。装箱是[值类型\(C# 参考\)](#)到 **object** 类型或到此值类型所实现的任何接口类型的隐式转换。对值类型装箱会在堆中分配一个对象实例，并将该值复制到新的对象中。

请看以下值类型变量的声明：

C#

```
int i = 123;
```

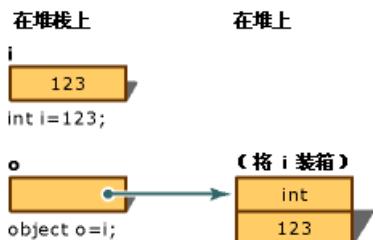
以下语句对变量 **i** 隐式应用装箱操作：

C#

```
object o = i; // implicit boxing
```

此语句的结果是在堆栈上创建对象引用 **o**，而在堆上则引用 **int** 类型的值。该值是赋给变量 **i** 的值类型值的一个副本。下图说明了两个变量 **i** 和 **o** 之间的差异。

装箱转换



还可以像下面的示例一样显式执行装箱，不过显式装箱从来不是必需的：

C#

```
int i = 123;
object o = (object)i; // explicit boxing
```

示例

此示例通过装箱将整数变量 **i** 转换为对象 **o**。这样，存储在变量 **i** 中的值就从 123 更改为 456。该示例表明原始值类型和装箱的对象使用不同的内存位置，因此能够存储不同的值。

C#

```
class TestBoxing
{
    static void Main()
    {
        int i = 123;
        object o = i; // implicit boxing

        i = 456; // change the contents of i

        System.Console.WriteLine("The value-type value = {0}", i);
        System.Console.WriteLine("The object-type value = {0}", o);
    }
}
```

输出

The value-type value = 456

The object-type value = 123

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 4.3.1 装箱转换

请参见

概念

[C# 编程指南](#)

[装箱和取消装箱\(C# 编程指南\)](#)

[取消装箱转换\(C# 编程指南\)](#)

[已装箱的值类型](#)

取消装箱转换(C# 编程指南)

取消装箱是从 `object` 类型到值类型或从接口类型到实现该接口的值类型的显式转换。取消装箱操作包括：

- 检查对象实例，确保它是给定值类型的一个装箱值。
- 将该值从实例复制到值类型变量中。

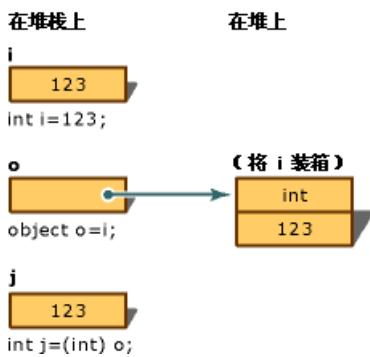
以下语句同时说明了装箱和取消装箱操作：

C#

```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;  // unboxing
```

下图显示了以上语句的结果。

取消装箱转换



要在运行时成功取消装箱值类型，被取消装箱的项必须是对一个对象的引用，该对象是先前通过装箱该值类型的实例创建的。尝试对 `null` 或对不兼容值类型的引用进行取消装箱操作，将导致 `InvalidCastException`。

示例

下面的示例演示无效的取消装箱及引发的 `InvalidCastException`。使用 `try` 和 `catch`，在发生错误时显示错误信息。

C#

```
class TestUnboxing
{
    static void Main()
    {
        int i = 123;
        object o = i; // implicit boxing

        try
        {
            int j = (short)o; // attempt to unbox

            System.Console.WriteLine("Unboxing OK.");
        }
        catch (System.InvalidCastException e)
        {
            System.Console.WriteLine("{0} Error: Incorrect unboxing.", e.Message);
        }
    }
}
```

输出

Specified cast is not valid. Error: Incorrect unboxing.

如果将下列语句：

```
int j = (short) o;
```

更改为：

```
int j = (int) o;
```

将执行转换，并将得到以下输出：

Unboxing OK.

请参见

概念

[C# 编程指南](#)

[装箱和取消装箱\(C# 编程指南\)](#)

[装箱转换\(C# 编程指南\)](#)

数组 (C# 编程指南)

数组是一种数据结构，它包含若干相同类型的变量。数组是使用类型声明的：

```
type[] arrayName;
```

下面的示例创建一维、多维和交错数组：

C#

```
class TestArraysClass
{
    static void Main()
    {
        // Declare a single-dimensional array
        int[] array1 = new int[5];

        // Declare and set array element values
        int[] array2 = new int[] { 1, 3, 5, 7, 9 };

        // Alternative syntax
        int[] array3 = { 1, 2, 3, 4, 5, 6 };

        // Declare a two dimensional array
        int[,] multiDimensionalArray1 = new int[2, 3];

        // Declare and set array element values
        int[,] multiDimensionalArray2 = { { 1, 2, 3 }, { 4, 5, 6 } };

        // Declare a jagged array
        int[][] jaggedArray = new int[6][];

        // Set the values of the first array in the jagged array structure
        jaggedArray[0] = new int[4] { 1, 2, 3, 4 };
    }
}
```

数组概述

数组具有以下属性：

- 数组可以是[一维](#)、[多维](#)或[交错](#)的。
- 数值数组元素的默认值设置为零，而引用元素的默认值设置为 `null`。
- 交错数组是数组的数组，因此，它的元素是引用类型，初始化为 `null`。
- 数组的索引从零开始：具有 n 个元素的数组的索引是从 0 到 $n-1$ 。
- 数组元素可以是任何类型，包括数组类型。
- 数组类型是从抽象基类型 `Array` 派生的[引用类型](#)。由于此类型实现了 `IEnumerable` 和 `IEnumerable`，因此可以对 C# 中的所有数组使用 `foreach` 迭代。

相关章节

- [作为对象的数组 \(C# 编程指南\)](#)
- [对数组使用 foreach \(C# 编程指南\)](#)
- [将数组作为参数传递 \(C# 编程指南\)](#)
- [使用 ref 和 out 传递数组 \(C# 编程指南\)](#)
- [“数组”示例](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.8 数组
- 12 数组

[请参见](#)

[参考](#)

[数组使用指南](#)

[概念](#)

[C# 编程指南](#)

[集合类\(C# 编程指南\)](#)

[数组集合类型](#)

[通用类型系统中的数组](#)

作为对象的数组 (C# 编程指南)

在 C# 中，数组实际上是对象，而不只是像 C 和 C++ 中那样的可寻址连续内存区域。[Array](#) 是所有数组类型的抽象基类型。可以使用 **Array** 具有的属性以及其他类成员。这种用法的一个示例是使用 [Length](#) 属性来获取数组的长度。下面的代码将 `numbers` 数组的长度(为 5)赋给名为 `lengthOfNumbers` 的变量：

C#

```
int[] numbers = { 1, 2, 3, 4, 5 };
int lengthOfNumbers = numbers.Length;
```

System.Array 类提供了许多其他有用的方法和属性，用于排序、搜索和复制数组。

示例

此示例使用 [Rank](#) 属性来显示数组的维数。

C#

```
class TestArraysClass
{
    static void Main()
    {
        // Declare and initialize an array:
        int[,] theArray = new int[5, 10];
        System.Console.WriteLine("The array has {0} dimensions.", theArray.Rank);
    }
}
```

输出

The array has 2 dimensions.

请参见

参考

[一维数组 \(C# 编程指南\)](#)

[多维数组 \(C# 编程指南\)](#)

[交错数组 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

一维数组 (C# 编程指南)

可以如下例所示声明一个由 5 个整数组成的数组：

C#

```
int[] array = new int[5];
```

此数组包含从 `array[0]` 到 `array[4]` 的元素。`new` 运算符用于创建数组并将数组元素初始化为它们的默认值。在此例中，所有数组元素都初始化为零。

可以用相同的方式声明存储字符串元素的数组。例如：

C#

```
string[] stringArray = new string[6];
```

数组初始化

可以在声明数组时将其初始化，在这种情况下不需要级别说明符，因为级别说明符已经由初始化列表中的元素数据提供。例如：

C#

```
int[] array1 = new int[5] { 1, 3, 5, 7, 9 };
```

可以用相同的方式初始化字符串数组。下面声明一个字符串数组，其中每个数组元素用每天的名称初始化：

C#

```
string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

如果在声明数组时将其初始化，则可以使用下列快捷方式：

C#

```
int[] array2 = { 1, 3, 5, 7, 9 };
string[] weekDays2 = { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };
```

可以声明一个数组变量但不将其初始化，但在将数组分配给此变量时必须使用 `new` 运算符。例如：

C#

```
int[] array3;
array3 = new int[] { 1, 3, 5, 7, 9 }; // OK
//array3 = {1, 3, 5, 7, 9}; // Error
```

值类型数组和引用类型数组

请看下列数组声明：

C#

```
SomeType[] array4 = new SomeType[10];
```

该语句的结果取决于 `SomeType` 是值类型还是引用类型。如果是值类型，则该语句将创建一个由 10 个 `SomeType` 类型的实例组成的数组。如果 `SomeType` 是引用类型，则该语句将创建一个由 10 个元素组成的数组，其中每个元素都初始化为空引用。

有关值类型和引用类型的更多信息，请参见[类型 \(C# 参考\)](#)。

[请参见](#)

[参考](#)

[多维数组\(C# 编程指南\)](#)

[交错数组\(C# 编程指南\)](#)

[Array](#)

[概念](#)

[C# 编程指南](#)

[数组\(C# 编程指南\)](#)

多维数组(C# 编程指南)

数组可以具有多个维度。例如，下列声明创建一个四行两列的二维数组：

C#

```
int[,] array = new int[4, 2];
```

另外，下列声明创建一个三维(4、2 和 3)数组：

C#

```
int[, ,] array1 = new int[4, 2, 3];
```

数组初始化

可以在声明数组时将其初始化，如下例所示：

C#

```
int[,] array2D = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
int[, ,] array3D = new int[,,] { { { 1, 2, 3 } }, { { 4, 5, 6 } } };
```

也可以初始化数组但不指定级别：

C#

```
int[,] array4 = { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } };
```

如果选择声明一个数组变量但不将其初始化，必须使用 **new** 运算符将一个数组分配给此变量。例如：

C#

```
int[,] array5;
array5 = new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } }; // OK
//array5 = {{1,2}, {3,4}, {5,6}, {7,8}}; // Error
```

也可以给数组元素赋值，例如：

C#

```
array5[2, 1] = 25;
```

下面的代码示例将数组变量初始化为默认值(交错数组除外)：

C#

```
int[,] array6 = new int[10, 10];
```

请参见

参考

[一维数组\(C# 编程指南\)](#)

[交错数组\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[数组\(C# 编程指南\)](#)

交错数组 (C# 编程指南)

交错数组是元素为数组的数组。交错数组元素的维度和大小可以不同。交错数组有时称为“数组的数组”。以下示例说明如何声明、初始化和访问交错数组。

下面声明一个由三个元素组成的一维数组，其中每个元素都是一个一维整数数组：

C#

```
int[][] jaggedArray = new int[3][];
```

必须初始化 `jaggedArray` 的元素后才可以使用它。可以如下例所示初始化该元素：

C#

```
jaggedArray[0] = new int[5];
jaggedArray[1] = new int[4];
jaggedArray[2] = new int[2];
```

每个元素都是一个一维整数数组。第一个元素是由 5 个整数组成的数组，第二个是由 4 个整数组成的数组，而第三个是由 2 个整数组成的数组。

也可以使用初始值设定项用值填充数组元素，在这种情况下不需要数组大小。例如：

C#

```
jaggedArray[0] = new int[] { 1, 3, 5, 7, 9 };
jaggedArray[1] = new int[] { 0, 2, 4, 6 };
jaggedArray[2] = new int[] { 11, 22 };
```

还可以在声明数组时将其初始化，如：

C#

```
int[][] jaggedArray2 = new int[][]
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

可以使用下面的速记格式。请注意：不能从元素初始化中省略 `new` 运算符，因为不存在元素的默认初始化：

C#

```
int[][] jaggedArray3 =
{
    new int[] {1,3,5,7,9},
    new int[] {0,2,4,6},
    new int[] {11,22}
};
```

交错数组是数组的数组，因此其元素是引用类型并初始化为 `null`。

可以如下例所示访问个别数组元素：

C#

```
// Assign 77 to the second element ([1]) of the first array ([0]):
jaggedArray3[0][1] = 77;
```

```
// Assign 88 to the second element ([1]) of the third array ([2]):  
jaggedArray3[2][1] = 88;
```

可以混合使用交错数组和多维数组。下面声明和初始化一个一维交错数组，该数组包含大小不同的二维数组元素：

C#

```
int[][] jaggedArray4 = new int[3][,]  
{  
    new int[,] { {1,3}, {5,7} },  
    new int[,] { {0,2}, {4,6}, {8,10} },  
    new int[,] { {11,22}, {99,88}, {0,9} }  
};
```

可以如本例所示访问个别元素，该示例显示第一个数组的元素 [1,0] 的值(值为 5)：

C#

```
System.Console.WriteLine("{0}", jaggedArray4[0][1, 0]);
```

方法 `Length` 返回包含在交错数组中的数组的数目。例如，假定您已声明了前一个数组，则此行：

C#

```
System.Console.WriteLine(jaggedArray4.Length);
```

将返回值 3。

示例

本例生成一个数组，该数组的元素为数组自身。每一个数组元素都有不同的大小。

C#

```
class ArrayTest  
{  
    static void Main()  
    {  
        // Declare the array of two elements:  
        int[][] arr = new int[2][];  
  
        // Initialize the elements:  
        arr[0] = new int[5] { 1, 3, 5, 7, 9 };  
        arr[1] = new int[4] { 2, 4, 6, 8 };  
  
        // Display the array elements:  
        for (int i = 0; i < arr.Length; i++)  
        {  
            System.Console.Write("Element({0}): ", i);  
  
            for (int j = 0; j < arr[i].Length; j++)  
            {  
                System.Console.Write("{0}{1}", arr[i][j], j == (arr[i].Length - 1) ? "" : "  
");  
            }  
            System.Console.WriteLine();  
        }  
    }  
}
```

输出

```
Element(0): 1 3 5 7 9
```

```
Element(1): 2 4 6 8
```

请参见

参考

[一维数组 \(C# 编程指南\)](#)

[多维数组 \(C# 编程指南\)](#)

[Array](#)

概念

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

对数组使用 foreach(C# 编程指南)

C# 还提供 `foreach` 语句。该语句提供一种简单、明了的方法来循环访问数组的元素。例如，下面的代码创建一个名为 `numbers` 的数组，并用 `foreach` 语句循环访问该数组：

C#

```
int[] numbers = { 4, 5, 6, 1, 2, 3, -2, -1, 0 };
foreach (int i in numbers)
{
    System.Console.WriteLine(i);
}
```

由于有了多维数组，可以使用相同方法来循环访问元素，例如：

C#

```
int[,] numbers2D = new int[3, 2] { { 9, 99 }, { 3, 33 }, { 5, 55 } };
foreach (int i in numbers2D)
{
    System.Console.Write("{0} ", i);
}
```

该示例的输出为：

9 99 3 33 5 55

然而，对于多维数组，使用嵌套的 `for` 循环可以更好地控制数组元素。

请参见

参考

[一维数组\(C# 编程指南\)](#)

[多维数组\(C# 编程指南\)](#)

[交错数组\(C# 编程指南\)](#)

[Array](#)

概念

[C# 编程指南](#)

[数组\(C# 编程指南\)](#)

将数组作为参数传递(C# 编程指南)

数组可作为参数传递给方法。因为数组是引用类型，所以方法可以更改元素的值。

将一维数组作为参数传递

可以将初始化的一维数组传递给方法。例如：

C#

```
PrintArray(theArray);
```

上面的行中调用的方法可定义为：

C#

```
void PrintArray(int[] arr)
{
    // method code
}
```

也可以在一个步骤中初始化并传递新数组。例如：

C#

```
PrintArray(new int[] { 1, 3, 5, 7, 9 });
```

示例 1

在下例中，初始化一个字符串数组并将其作为参数传递给 `PrintArray` 方法（该数组的元素显示在此方法中）：

C#

```
class ArrayClass
{
    static void PrintArray(string[] arr)
    {
        for (int i = 0; i < arr.Length; i++)
        {
            System.Console.Write(arr[i] + "{0}", i < arr.Length - 1 ? " " : "");
        }
        System.Console.WriteLine();
    }

    static void Main()
    {
        // Declare and initialize an array:
        string[] weekDays = new string[] { "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat" };

        // Pass the array as a parameter:
        PrintArray(weekDays);
    }
}
```

输出 1

Sun Mon Tue Wed Thu Fri Sat

将多维数组作为参数传递

可以将初始化的多维数组传递给方法。例如，如果 `theArray` 是二维数组：

C#

```
PrintArray(theArray);
```

上面的行中调用的方法可定义为：

C#

```
void PrintArray(int[,] arr)
{
    // method code
}
```

也可以在一个步骤中初始化并传递新数组。例如：

C#

```
PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });
```

示例 2

在此示例中，初始化一个二维数组并将其传递给 PrintArray 方法（该数组的元素显示在此方法中）。

C#

```
class ArrayClass2D
{
    static void PrintArray(int[,] arr)
    {
        // Display the array elements:
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 2; j++)
            {
                System.Console.WriteLine("Element({0},{1})={2}", i, j, arr[i, j]);
            }
        }
    }
    static void Main()
    {
        // Pass the array as a parameter:
        PrintArray(new int[,] { { 1, 2 }, { 3, 4 }, { 5, 6 }, { 7, 8 } });
    }
}
```

输出 2

```
Element(0,0)=1
Element(0,1)=2
Element(1,0)=3
Element(1,1)=4
Element(2,0)=5
Element(2,1)=6
Element(3,0)=7
Element(3,1)=8
```

请参见

参考

[一维数组\(C# 编程指南\)](#)

[多维数组\(C# 编程指南\)](#)

[交错数组\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

使用 **ref** 和 **out** 传递数组 (C# 编程指南)

与所有的 **out** 参数一样，在使用数组类型的 **out** 参数前必须先为其赋值，即必须由被调用方为其赋值。例如：

C#

```
static void TestMethod1(out int[] arr)
{
    arr = new int[10]; // definite assignment of arr
}
```

与所有的 **ref** 参数一样，数组类型的 **ref** 参数必须由调用方明确赋值。因此不需要由接受方明确赋值。可以将数组类型的 **ref** 参数更改为调用的结果。例如，可以为数组赋以 **null** 值，或将其初始化为另一个数组。例如：

C#

```
static void TestMethod2(ref int[] arr)
{
    arr = new int[10]; // arr initialized to a different array
}
```

下面的两个示例说明 **out** 与 **ref** 在将数组传递给方法时的用法差异。

示例 1

在此例中，在调用方(`Main`方法)中声明数组 `theArray`，并在 `FillArray` 方法中初始化此数组。然后将数组元素返回调用方并显示。

C#

```
class TestOut
{
    static void FillArray(out int[] arr)
    {
        // Initialize the array:
        arr = new int[5] { 1, 2, 3, 4, 5 };
    }

    static void Main()
    {
        int[] theArray; // Initialization is not required

        // Pass the array to the callee using out:
        FillArray(out theArray);

        // Display the array elements:
        System.Console.WriteLine("Array elements are:");
        for (int i = 0; i < theArray.Length; i++)
        {
            System.Console.Write(theArray[i] + " ");
        }
    }
}
```

输出 1

Array elements are:

1 2 3 4 5

示例 2

在此例中，在调用方(`Main`方法)中初始化数组 `theArray`，并通过使用 **ref** 参数将其传递给 `FillArray` 方法。在 `FillArray` 方法中更新某些数组元素。然后将数组元素返回调用方并显示。

C#

```
class TestRef
{
    static void FillArray(ref int[] arr)
    {
        // Create the array on demand:
        if (arr == null)
        {
            arr = new int[10];
        }
        // Fill the array:
        arr[0] = 1111;
        arr[4] = 5555;
    }

    static void Main()
    {
        // Initialize the array:
        int[] theArray = { 1, 2, 3, 4, 5 };

        // Pass the array using ref:
        FillArray(ref theArray);

        // Display the updated array:
        System.Console.WriteLine("Array elements are:");
        for (int i = 0; i < theArray.Length; i++)
        {
            System.Console.Write(theArray[i] + " ");
        }
    }
}
```

输出 2

Array elements are:

1111 2 3 4 5555

请参见

参考

[一维数组\(C# 编程指南\)](#)

[多维数组\(C# 编程指南\)](#)

[交错数组\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[数组\(C# 编程指南\)](#)

字符串 (C# 编程指南)

以下章节讨论 **string** 数据类型，它是 [String](#) 类的别名。

本节内容

[使用字符串 \(C# 编程指南\)](#)

[如何：使用 Split 方法分析字符串 \(C# 编程指南\)](#)

[如何：使用字符串方法搜索字符串 \(C# 编程指南\)](#)

[如何：使用正则表达式搜索字符串 \(C# 编程指南\)](#)

[如何：联接多个字符串 \(C# 编程指南\)](#)

[如何：修改字符串内容 \(C# 编程指南\)](#)

相关章节

[基本字符串操作](#)

[比较字符串](#)

请参见

[概念](#)

[C# 编程指南](#)

使用字符串 (C# 编程指南)

C# 字符串是使用 **string** 关键字声明的一个字符数组。字符串是使用引号声明的，如下例所示：

C#

```
string s = "Hello, World!";
```

您可以提取子字符串和连接字符串，如下所示：

C#

```
string s1 = "orange";
string s2 = "red";

s1 += s2;
System.Console.WriteLine(s1); // outputs "orangered"

s1 = s1.Substring(2, 5);
System.Console.WriteLine(s1); // outputs "anger"
```

字符串对象是“不可变的”，即它们一旦创建就无法更改。对字符串进行操作的方法实际上返回的是新的字符串对象。在前面的示例中，将 `s1` 和 `s2` 的内容连接起来以构成一个字符串时，包含 “orange” 和 “red” 的两个字符串均保持不变。`+=` 运算符会创建一个包含组合内容的新字符串。结果是 `s1` 现在引用一个完全不同的字符串。只包含 “orange” 的字符串仍然存在，但连接 `s1` 后将不再被引用。

注意

创建字符串引用时要谨慎从事。如果您创建了一个字符串引用，然后“修改”了该字符串，则该引用会继续指向原始对象，而不是修改字符串时创建的新对象。下面的代码阐释了这个危险：

```
string s1 = "Hello";
string s2 = s1;
s1 += " and goodbye.";
Console.WriteLine(s2); //outputs "Hello"
```

因为修改字符串的操作涉及创建新字符串对象，出于性能方面的考虑，大量的串联或所涉及其他字符串操作应通过 [StringBuilder](#) 类来执行，如下所示：

C#

```
System.Text.StringBuilder sb = new System.Text.StringBuilder();
sb.Append("one ");
sb.Append("two ");
sb.Append("three");
string str = sb.ToString();
```

[StringBuilder](#) 类在“使用 Stringbuilder”一节中有进一步的讨论。

使用字符串

转义符

字符串中可以包含转义符，如“\n”（新行）和“\t”（制表符）。行：

C#

```
string hello = "Hello\nWorld!";
```

等同于：

```
Hello
```

```
World!
```

如果希望包含反斜杠，则它前面必须还有另一个反斜杠。下面的字符串：

```
C#
```

```
string filePath = "\\\\"My Documents\\\";
```

实际上等同于：

```
\My Documents\
```

@ 符号

@ 符号会告知字符串构造函数忽略转义符和分行符。因此，以下两个字符串是完全相同的：

```
C#
```

```
string p1 = "\\\\"My Documents\\\"My Files\\\";  
string p2 = @"\\My Documents\My Files\";
```

ToString()

如同所有从 `Object` 派生的对象一样，字符串也提供了 `ToString` 方法，用于将值转换为字符串。此方法可用于将数值转换为字符串，如下所示：

```
C#
```

```
int year = 1999;  
string msg = "Eve was born in " + year.ToString();  
System.Console.WriteLine(msg); // outputs "Eve was born in 1999"
```

访问各个字符

字符串中所包含的各个字符可以使用以下方法进行访问，如 `SubString()`、`Replace()`、`Split()` 和 `Trim()`。

```
C#
```

```
string s3 = "Visual C# Express";  
  
System.Console.WriteLine(s3.Substring(7, 2)); // outputs "C#"  
System.Console.WriteLine(s3.Replace("C#", "Basic")); // outputs "Visual Basic Express"
```

也可以将字符复制到字符数组，如下所示：

```
C#
```

```
string s4 = "Hello, World";  
char[] arr = s4.ToCharArray(0, s4.Length);  
  
foreach (char c in arr)  
{  
    System.Console.Write(c); // outputs "Hello, World"  
}
```

可以用索引访问字符串中的各个字符，如下所示：

```
C#
```

```
string s5 = "Printing backwards";  
  
for (int i = 0; i < s5.Length; i++)
```

```
{  
    System.Console.WriteLine(s5[s5.Length - i - 1]); // outputs "sdrawkcab gnitnirP"  
}
```

更改大小写

若要将字符串中的字母更改为大写或小写，可以使用 **ToUpper()** 或 **ToLower()**，如下所示：

C#

```
string s6 = "Battle of Hastings, 1066";  
  
System.Console.WriteLine(s6.ToUpper()); // outputs "BATTLE OF HASTINGS 1066"  
System.Console.WriteLine(s6.ToLower()); // outputs "battle of hastings 1066"
```

比较

比较两个字符串的最简单方法是使用 **==** 和 **!=** 运算符，执行区分大小写的比较。

C#

```
string color1 = "red";  
string color2 = "green";  
string color3 = "red";  
  
if (color1 == color3)  
{  
    System.Console.WriteLine("Equal");  
}  
if (color1 != color2)  
{  
    System.Console.WriteLine("Not equal");  
}
```

字符串对象也有一个 **CompareTo()** 方法，它根据某个字符串是否小于 (<) 或大于 (>) 另一个，返回一个整数值。比较字符串时使用 Unicode 值，小写的值小于大写的值。

C#

```
string s7 = "ABC";  
string s8 = "abc";  
  
if (s7.CompareTo(s8) > 0)  
{  
    System.Console.WriteLine("Greater-than");  
}  
else  
{  
    System.Console.WriteLine("Less-than");  
}
```

若要在字符串中搜索另一个字符串，可以使用 **IndexOf()**。如果未找到搜索字符串，**IndexOf()** 返回 -1；否则，返回它出现的第一个位置的索引（从零开始）。

C#

```
string s9 = "Battle of Hastings, 1066";  
  
System.Console.WriteLine(s9.IndexOf("Hastings")); // outputs 10  
System.Console.WriteLine(s9.IndexOf("1967")); // outputs -1
```

将字符串拆分为子字符串

将字符串拆分为子字符串(如将句子拆分为各个单词)是一个常见的编程任务。**Split()** 方法使用分隔符(如空格字符)**char** 数组，并返回一个子字符串数组。您可以使用 **foreach** 访问此数组，如下所示：

C#

```
char[] delimiter = new char[] { ' ' };
string s10 = "The cat sat on the mat.";
foreach (string substr in s10.Split(delimiter))
{
    System.Console.WriteLine(substr);
}
```

此代码将在单独的行上输出每个单词，如下所示：

```
The
cat
sat
on
the
mat.
```

Null 字符串和空字符串

空字符串是不包含字符的 **System.String** 对象的实例。在各种编程方案中经常会使用空字符串表示空白文本字段。可以对空字符串调用方法，因为它们是有效的 **System.String** 对象。空字符串可按如下方式初始化：

```
string s = "";
```

相反，null 字符串并不涉及 **System.String** 对象的实例，任何对 null 字符串调用方法的尝试都会生成 **NullReferenceException**。但是，可以在串联和比较操作中将 null 字符串与其他字符串一起使用。下面的示例阐释了引用 null 字符串导致引发异常的情形以及并不导致引发异常的情形：

```
string str = "hello";
string nullStr = null;
string emptyStr = "";

string tempStr = str + nullStr; // tempStr = "hello"
bool b = (emptyStr == nullStr); // b = false;
emptyStr + nullStr = ""; // creates a new empty string
int I = nullStr.Length; // throws NullReferenceException
```

使用 **StringBuilder**

StringBuilder 类创建了一个字符串缓冲区，用于在程序执行大量字符串操作时提供更好的性能。**StringBuilder** 字符串还允许您重新分配个别字符，这些字符是内置字符串数据类型所不支持的。例如，此代码在不创建新字符串的情况下更改了一个字符串的内容：

C#

```
System.Text.StringBuilder sb = new System.Text.StringBuilder("Rat: the ideal pet");
sb[0] = 'C';
System.Console.WriteLine(sb.ToString());
System.Console.ReadLine();
```

在本示例中，**StringBuilder** 对象用于从一组数值类型中创建字符串：

C#

```
class TestStringBuilder
{
    static void Main()
    {
```

```
System.Text.StringBuilder sb = new System.Text.StringBuilder();

// Create a string composed of numbers 0 - 9
for (int i = 0; i < 10; i++)
{
    sb.Append(i.ToString());
}
System.Console.WriteLine(sb); // displays 0123456789

// Copy one character of the string (not possible with a System.String)
sb[0] = sb[9];

System.Console.WriteLine(sb); // displays 9123456789
}
```

更多信息

在 C# 程序员参考中：

- [string\(C# 参考\)](#)

请参见

任务

[如何：使用正则表达式搜索字符串 \(C# 编程指南\)](#)

[如何：使用字符串方法搜索字符串 \(C# 编程指南\)](#)

[如何：使用 Split 方法分析字符串 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[分析字符串](#)

如何：使用 Split 方法分析字符串 (C# 编程指南)

下面的代码示例演示如何使用 [System.String.Split](#) 方法分析字符串。此方法返回一个字符串数组，其中每个元素是一个单词。作为输入，[Split](#) 采用一个字符数组指示哪些字符被用作分隔符。本示例中使用了空格、逗号、句点、冒号和制表符。一个含有这些分隔符的数组被传递给 [Split](#)，并使用结果字符串数组分别显示句子中的每个单词。

示例

C#

```
class TestStringSplit
{
    static void Main()
    {
        char[] delimiterChars = { ' ', ',', '.', ':', '\t' };

        string text = "one\ttwo three:four,five six seven";
        System.Console.WriteLine("Original text: '{0}'", text);

        string[] words = text.Split(delimiterChars);
        System.Console.WriteLine("{0} words in text:", words.Length);

        foreach (string s in words)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

输出

```
Original text: 'one      two three:four,five six seven'
7 words in text:
one
two
three
four
five
six
seven
```

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[字符串 \(C# 编程指南\)](#)

[.NET Framework 正则表达式](#)

[使用 C++ 进行 .NET 编程](#)

如何：使用字符串方法搜索字符串 (C# 编程指南)

`string` 类型(它是 `System.String` 类的别名)为搜索字符串的内容提供了许多有用的方法。下面的示例使用 `IndexOf`、`LastIndexOf`、`StartsWith` 和 `EndsWith` 方法。

示例

C#

```
class StringSearch
{
    static void Main()
    {
        string str = "A silly sentence used for silly purposes.";
        System.Console.WriteLine('{0}', str);

        bool test1 = str.StartsWith("a silly");
        System.Console.WriteLine("starts with 'a silly'? {0}", test1);

        bool test2 = str.StartsWith("a silly", System.StringComparison.OrdinalIgnoreCase);
        System.Console.WriteLine("starts with 'a silly'? {0} (ignoring case)", test2);

        bool test3 = str.EndsWith(".");
        System.Console.WriteLine("ends with '.'? {0}", test3);

        int first = str.IndexOf("silly");
        int last = str.LastIndexOf("silly");
        string str2 = str.Substring(first, last - first);
        System.Console.WriteLine("between two 'silly' words: '{0}'", str2);
    }
}
```

输出

```
'A silly sentence used for silly purposes.'
starts with 'a silly'? False
starts with 'a silly'? True (ignore case)
ends with '.'? True
between two 'silly' words: 'silly sentence used for '
```

请参见

概念

[C# 编程指南](#)

其他资源

[字符串 \(C# 编程指南\)](#)

如何：使用正则表达式搜索字符串 (C# 编程指南)

可以使用 [System.Text.RegularExpressions.Regex](#) 类搜索字符串。这些搜索可以涵盖从非常简单到全面使用正则表达式的复杂范围。以下是使用 **Regex** 类搜索字符串的两个示例。有关更多信息，请参见 [.NET Framework 正则表达式](#)。

示例

以下代码是一个控制台应用程序，用于对数组中的字符串执行简单的不区分大小写的搜索。给定要搜索的字符串和包含搜索模式的字符串后，静态方法

[System.Text.RegularExpressions.Regex.IsMatch\(System.String, System.String, System.Text.RegularExpressions.RegexOptions\)](#) 执行搜索。在本例中，使用第三个参数指示忽略大小写。有关更多信息，请参见 [System.Text.RegularExpressions.RegexOptions](#)。

C#

```
class TestRegularExpressions
{
    static void Main()
    {
        string[] sentences =
        {
            "cow over the moon",
            "Betsy the Cow",
            "cowering in the corner",
            "no match here"
        };

        string sPattern = "cow";

        foreach (string s in sentences)
        {
            System.Console.WriteLine("{0,24}", s);

            if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern, System.Text.RegularExpressions.RegexOptions.IgnoreCase))
            {
                System.Console.WriteLine("  (match for '{0}' found)", sPattern);
            }
            else
            {
                System.Console.WriteLine();
            }
        }
    }
}
```

输出

```
cow over the moon  (match for 'cow' found)
Betsy the Cow  (match for 'cow' found)
cowering in the corner  (match for 'cow' found)
no match here
```

以下代码是一个控制台应用程序，此程序使用正则表达式验证数组中每个字符串的格式。验证要求每个字符串具有电话号码的形式，即用短划线将数字分成三组，前两组各包含三个数字，第三组包含四个数字。这是通过正则表达式 `^\d{3}-\d{3}-\d{4}$` 完成的。有关更多信息，请参见 [正则表达式语言元素](#)。

C#

```
class TestRegularExpressionValidation
{
    static void Main()
    {
        string[] numbers =
        {
            "123-456-7890",
```

```
    "444-234-22450",
    "690-203-6578",
    "146-893-232",
    "146-839-2322",
    "4007-295-1111",
    "407-295-1111",
    "407-2-5555",
};

string sPattern = "^\\d{3}-\\d{3}-\\d{4}$";

foreach (string s in numbers)
{
    System.Console.WriteLine("{0,14}", s);

    if (System.Text.RegularExpressions.Regex.IsMatch(s, sPattern))
    {
        System.Console.WriteLine(" - valid");
    }
    else
    {
        System.Console.WriteLine(" - invalid");
    }
}
}
```

输出

```
123-456-7890 - valid
444-234-22450 - invalid
690-203-6578 - valid
146-893-232 - invalid
146-839-2322 - valid
4007-295-1111 - invalid
407-295-1111 - valid
407-2-5555 - invalid
```

请参见

概念

[C# 编程指南](#)

其他资源

[字符串 \(C# 编程指南\)](#)

如何：联接多个字符串 (C# 编程指南)

有两种联接多个字符串的方法：使用 [String](#) 类重载的 + 运算符，以及使用 [StringBuilder](#) 类。+ 运算符使用方便，有助于生成直观的代码，但必须连续使用；每使用一次该运算符就创建一个新的字符串，因此将多个运算符串联在一起效率不高。例如：

C#

```
string two = "two";
string str = "one " + two + " three";
System.Console.WriteLine(str);
```

尽管在代码中只出现了四个字符串，三个字符串联接在一起，最后一个字符串包含全部三个字符串，但总共要创建五个字符串，因为首先要将前两个字符串联接，创建一个包含前两个字符串的字符串。第三个字符串是单独追加的，形成存储在 str 中的最终字符串。

也可以使用 [StringBuilder](#) 类将每个字符串添加到一个对象中，然后由该对象通过一个步骤创建最终的字符串。下面的示例对此策略进行了演示。

示例

下面的代码使用 [StringBuilder](#) 类的 Append 方法来联接三个字符串，从而避免了串联多个 + 运算符的弊端。

C#

```
class StringBuilderTest
{
    static void Main()
    {
        string two = "two";

        System.Text.StringBuilder sb = new System.Text.StringBuilder();
        sb.Append("one ");
        sb.Append(two);
        sb.Append(" three");
        System.Console.WriteLine(sb.ToString());

        string str = sb.ToString();
        System.Console.WriteLine(str);
    }
}
```

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[字符串 \(C# 编程指南\)](#)

如何：修改字符串内容 (C# 编程指南)

字符串是不可变的，因此不能修改字符串的内容。但是，可以将字符串的内容提取到非不可变的窗体中，并对其进行修改，以形成新的字符串实例。

示例

下面的示例使用 [ToCharArray](#) 方法来将字符串的内容提取到 `char` 类型的数组中。然后修改此数组中的某些元素。之后，使用 `char` 数组创建新的字符串实例。

C#

```
class ModifyStrings
{
    static void Main()
    {
        string str = "The quick brown fox jumped over the fence";
        System.Console.WriteLine(str);

        char[] chars = str.ToCharArray();
        int animalIndex = str.IndexOf("fox");
        if (animalIndex != -1)
        {
            chars[animalIndex++] = 'c';
            chars[animalIndex++] = 'a';
            chars[animalIndex] = 't';
        }

        string str2 = new string(chars);
        System.Console.WriteLine(str2);
    }
}
```

输出

```
The quick brown fox jumped over the fence
The quick brown cat jumped over the fence
```

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[字符串 \(C# 编程指南\)](#)

语句、表达式和运算符(C# 编程指南)

本节包含关于构成 C# 程序的基本元素的信息。构成应用程序的 C# 代码由包含 C# 关键字、表达式和运算符的语句组成。

有关更多信息，请参见：

- [语句\(C# 编程指南\)](#)
- [表达式\(C# 编程指南\)](#)
- [运算符\(C# 编程指南\)](#)
- [可重载运算符\(C# 编程指南\)](#)
- [转换运算符\(C# 编程指南\)](#)
 - [使用转换运算符\(C# 编程指南\)](#)
 - [如何：在结构间实现用户定义的转换\(C# 编程指南\)](#)
- [如何：使用运算符重载创建复数类\(C# 编程指南\)](#)
- [Equals\(\) 和运算符 == 的重载准则\(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1.4 表达式](#)
- [1.5 语句](#)
- [1.6.6.5 运算符](#)
- [5.3.3 确定明确赋值的细则](#)
- [7 表达式](#)
- [7.2 运算符](#)
- [8 语句](#)

请参见

参考

[强制转换\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

语句(C# 编程指南)

语句是构造所有 C# 程序的过程构造块。语句可以声明局部变量或常数，调用方法，创建对象或将值赋予变量、属性或字段。控制语句可以创建循环，如 `for` 循环，也可以进行判断并分支到新的代码块，如 `if` 或 `switch` 语句。语句通常以分号终止。有关更多信息，请参见[语句类型\(C# 参考\)](#)。

由大括号括起来的一系列语句构成代码块。方法体是代码块的一个示例。代码块通常出现在控制语句之后。在代码块中声明的变量或常数只可用于同一代码块中的语句。例如，下面的代码演示在控制语句之后出现的一个方法块和一个代码块：

C#

```
bool IsPositive(int number)
{
    if (number > 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

C# 中的语句通常包含表达式。C# 中的表达式是一个包含文本值、简单名称或运算符及其操作数的代码段。大多数常用表达式在计算时都将产生文本值、变量、对象属性或对象索引器访问。只要从表达式中识别变量、对象属性或对象索引器访问，该项的值都将用作表达式的值。在 C# 中，表达式可以置于需要值或对象的任意位置，条件是表达式最终的计算结果必须为所需的类型。

有些表达式的计算结果为命名空间、类型、方法组或事件访问。这些具有特殊用途的表达式只在某些情况下（通常是作为较大的表达式的一部分时）有效，如果使用不正确，则将产生编译器错误。

相关章节

- [语句类型\(C# 参考\)](#)
- [表达式\(C# 编程指南\)](#)
- [运算符\(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.5 语句
- 7 表达式
- 8 语句

请参见

概念

[C# 编程指南](#)

表达式 (C# 编程指南)

表达式是可以计算且结果为单个值、对象、方法或命名空间的代码片段。表达式可以包含文本值、方法调用、运算符及其操作数，或简单名称。简单名称可以是变量、类型成员、方法参数、命名空间或类型的名称。

表达式可以使用运算符，而运算符又可以将其他表达式用作参数，或者使用方法调用，而方法调用的参数又可以是其他方法调用，因此表达式既可以非常简单，也可以非常复杂。

文本和简单名称

最简单的两种表达式类型是文本和简单名称。文本是没有名称的常数值。例如，在下面的代码示例中，5 和 "Hello World" 都是文本值：

C#

```
int i = 5;
string s = "Hello World";
```

有关文本的更多信息，请参见[类型 \(C# 参考\)](#)。

在上面的示例中，i 和 s 都是用于标识局部变量的简单名称。在表达式中使用这些变量时，变量值检索后将用于表达式中。例如，在下面的代码示例中，当调用 DoWork 时，该方法默认情况下接收值"5"，且不能访问变量 var：

C#

```
int var = 5;
DoWork(var);
```

调用表达式

在下面的代码示例中，对 DoWork 的调用是另一种表达式，称为调用表达式。

C#

```
DoWork(var);
```

具体来说，调用方法即是一个方法调用表达式。方法调用要求使用方法的名称（如前面的示例中那样作为名称，或作为其他表达式的结果），后跟括号和任意方法参数。有关更多信息，请参见[方法 \(C# 编程指南\)](#)。委托调用使用委托的名称和括号内的方法参数。有关更多信息，请参见[委托 \(C# 编程指南\)](#)。如果方法返回值，则方法调用和委托调用的计算结果为该方法的返回值。返回 void 的方法不能替代表达式中的值。

备注

只要从表达式中识别到变量、对象属性或对象索引器访问，该项的值都会用作表达式的值。在 C# 中，只要表达式的最终计算结果是所需的类型，表达式就可以放置在任何需要值或对象的位置。

请参见

参考

[方法 \(C# 编程指南\)](#)

[运算符 \(C# 编程指南\)](#)

[数据类型 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[委托 \(C# 编程指南\)](#)

运算符 (C# 编程指南)

在 C# 中，运算符是术语或符号，它接受一个或多个称为操作数的表达式作为输入并返回值。接受一个操作数的运算符被称作一元运算符，例如增量运算符 `(++)` 或 `new`。接受两个操作数的运算符被称作二元运算符，例如算术运算符 `+`、`-`、`*`、`/`。条件运算符 `?:` 接受三个操作数，是 C# 中唯一的三元运算符。

下面的 C# 语句包含一个一元运算符和一个操作数。增量运算符 `++` 修改操作数 `y` 的值：

C#

```
y++;
```

下面的 C# 语句包含两个二元运算符，它们分别有两个操作数。赋值运算符 `=` 将一个整数 `y` 和一个表达式 `2 + 3` 作为操作数。表达式 `2 + 3` 本身包含加运算符，并使用整数值 `2` 和 `3` 作为操作数：

C#

```
y = 2 + 3;
```

操作数可以是任何大小、由任何数量的其他操作组成的有效表达式。

表达式中的运算符按照称为运算符优先级的特定顺序计算。下表根据运算符执行的操作类型将它们划分到不同的类别中。类别按优先级顺序列出。

基本	<code>x.y, f(x), a[x], x++, x--, new, typeof, checked, unchecked</code>
一元	<code>+、-、!、~、++x、--x、(T)x</code>
算术 -- 乘法	<code>*, /, %</code>
算术 -- 加法	<code>+, -</code>
移位	<code><<, >></code>
关系和类型检测	<code><, >, <=, >=, is, as</code>
相等	<code>==, !=</code>
逻辑(按优先级顺序)	<code>&, ^, </code>
条件(按优先级顺序)	<code>&&, , ?:</code>
赋值	<code>=, +=, -=, *=, /=, %=, &=, =, ^=, <<=, >>=</code>

当表达式中出现两个具有相同优先级的运算符时，它们根据结合性进行计算。左结合运算符按从左到右的顺序计算。例如，`x * y / z` 计算为 `(x * y) / z`。右结合运算符按从右到左的顺序计算。赋值运算符和三元运算符 (`?:`) 是右结合运算符。其他所有二元运算符都是左结合运算符。然而，C# 标准没有指定何时执行表达式中的增量指令的“设置”部分。例如，下面的代码示例的输出为 6：

C#

```
int num1 = 5;
num1++;
System.Console.WriteLine(num1);
```

而下面的代码示例的输出却是未定义的：

C#

```
int num2 = 5;
num2 = num2++; //not recommended
System.Console.WriteLine(num2);
```

因此，建议不要使用后一个示例。可以在表达式两侧使用括号来强制在计算其他任何表达式之前计算该表达式。例如， $2 + 3 * 2$ 正常情况下计算为 8。这是因为乘法运算符的优先级高于加法运算符。将该表达式写为 $(2 + 3) * 2$ 的形式，结果将是 10，因为它指示 C# 编译器必须在计算乘法运算符 * 之前计算加法运算符 +。

对于自定义的类和结构，您可以更改运算符的行为。此过程称为运算符重载。有关更多信息，请参见[可重载运算符\(C# 编程指南\)](#)。

相关章节

有关更多信息，请参见[运算符关键字\(C# 参考\)](#)和[C# 运算符](#)。

请参见

参考

[语句、表达式和运算符\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

可重载运算符 (C# 编程指南)

C# 允许用户定义的类型通过使用 `operator` 关键字定义静态成员函数来重载运算符。但不是所有的运算符都可被重载，下表列出了不能被重载的运算符：

运算符	可重载性
<code>+、-、!、~、++、--、true 和 false</code>	可以重载这些一元运算符。
<code>+, -, *, /, %, &, , ^, <<, >></code>	可以重载这些二进制运算符。
<code>==, !=, <, >, <=, >=</code>	比较运算符可以重载(但请参见本表后面的说明)。
<code>&&, </code>	条件逻辑运算符不能重载，但可使用能够重载的 <code>&</code> 和 <code> </code> 进行计算。
<code>[]</code>	不能重载数组索引运算符，但可定义索引器。
<code>()</code>	不能重载转换运算符，但可定义新的转换运算符(请参见 <code>explicit</code> 和 <code>implicit</code>)。
<code>+=, -=, *=, /=, %=, &=, =, ^=, <<=, >>=</code>	赋值运算符不能重载，但 <code>+=</code> 可使用 <code>+</code> 计算，等等。
<code>=、..、?:、->、new、is、sizeof 和 typeof</code>	不能重载这些运算符。

注意

比较运算符(如果重载)必须成对重载；也就是说，如果重载 `==`，也必须重载 `!=`。反之亦然，`<` 和 `>` 以及 `<=` 和 `>=` 同样如此。

若要在自定义类中重载运算符，您需要在该类中创建具有正确签名的方法。该方法必须命名为“operator X”，其中 X 是正在重载的运算符的名称或符号。一元运算符具有一个参数，二元运算符具有两个参数。在每种情况下，参数的类型必须与声明该运算符的类或结构的类型相同，如下面的示例所示：

C#

```
public static Complex operator +(Complex c1, Complex c2)
```

有关更多信息，请参见[如何：使用运算符重载创建复数类 \(C# 编程指南\)](#)。

请参见

参考

[语句、表达式和运算符 \(C# 编程指南\)](#)

[运算符 \(C# 编程指南\)](#)

[C# 运算符](#)

概念

[C# 编程指南](#)

转换运算符(C# 编程指南)

C# 允许程序员在类或结构上声明转换，以便类或结构与其他类或结构或者基本类型进行相互转换。转换的定义方法类似于运算符，并根据它们所转换到的类型命名。要转换的类型参数或转换结果的类型必须是(不能两者同时都是)包含类型。

C#

```
class SampleClass
{
    public static explicit operator SampleClass(int i)
    {
        SampleClass temp = new SampleClass();
        // code to convert from int to SampleClass...

        return temp;
    }
}
```

转换运算符概述

转换运算符具有以下特点：

- 声明为 **implicit** 的转换在需要时自动进行。
- 声明为 **explicit** 的转换需要调用强制转换。
- 所有转换都必须是 **static** 转换。

相关章节

更多信息：

- [使用转换运算符\(C# 编程指南\)](#)
- [转换运算符设计指南](#)
- [如何:在结构间实现用户定义的转换\(C# 编程指南\)](#)
- [explicit\(C# 参考\)](#)
- [implicit\(C# 参考\)](#)
- [static\(C# 参考\)](#)

请参见

概念

[C# 编程指南](#)

使用转换运算符 (C# 编程指南)

转换运算符可以是 **explicit**, 也可以是 **implicit**。隐式转换运算符更容易使用, 但是如果您希望运算符的用户能够意识到正在进行转换, 则显式运算符很有用。此主题演示了这两种类型。

示例 1

这是显式转换运算符的一个示例。此运算符将类型 **Byte** 转换为称为 **Digit** 的值类型。由于不是所有字节都可以转换为数字, 因此转换是显式的, 这意味着必须使用强制转换, 如 **Main** 方法所示。

C#

```
struct Digit
{
    byte value;

    public Digit(byte value) //constructor
    {
        if (value > 9)
        {
            throw new System.ArgumentException();
        }
        this.value = value;
    }

    public static explicit operator Digit(byte b) // explicit byte to digit conversion operator
    {
        Digit d = new Digit(b); // explicit conversion
        System.Console.WriteLine("conversion occurred");
        return d;
    }
}

class TestExplicitConversion
{
    static void Main()
    {
        try
        {
            byte b = 3;
            Digit d = (Digit)b; // explicit conversion
        }
        catch (System.Exception e)
        {
            System.Console.WriteLine("{0} Exception caught.", e);
        }
    }
}
```

输出 1

```
conversion occurred
```

示例 2

此示例通过定义用来撤消前一个示例所执行的操作的转换运算符, 来演示隐式转换运算符: 它将名为 **Digit** 的值类转换为整数 **Byte** 类型。由于任何数字都可以转换为 **Byte**, 因此没有必要一定让用户知道进行的转换。

C#

```
struct Digit
{
    byte value;
```

```
public Digit(byte value) //constructor
{
    if (value > 9)
    {
        throw new System.ArgumentException();
    }
    this.value = value;
}

public static implicit operator byte(Digit d) // implicit digit to byte conversion operator
{
    System.Console.WriteLine("conversion occurred");
    return d.value; // implicit conversion
}

class TestImplicitConversion
{
    static void Main()
    {
        Digit d = new Digit(3);
        byte b = d; // implicit conversion -- no cast needed
    }
}
```

输出 2

conversion occurred

请参见

概念

[C# 编程指南](#)

[转换运算符 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

如何:在结构间实现用户定义的转换(C# 编程指南)

本示例定义 RomanNumeral 和 BinaryNumeral 两个结构，并演示二者之间的转换。

示例

C#

```
struct RomanNumeral
{
    private int value;

    public RomanNumeral(int value) //constructor
    {
        this.value = value;
    }

    static public implicit operator RomanNumeral(int value)
    {
        return new RomanNumeral(value);
    }

    static public implicit operator RomanNumeral(BinaryNumeral binary)
    {
        return new RomanNumeral((int)binary);
    }

    static public explicit operator int(RomanNumeral roman)
    {
        return roman.value;
    }

    static public implicit operator string(RomanNumeral roman)
    {
        return ("Conversion not yet implemented");
    }
}

struct BinaryNumeral
{
    private int value;

    public BinaryNumeral(int value) //constructor
    {
        this.value = value;
    }

    static public implicit operator BinaryNumeral(int value)
    {
        return new BinaryNumeral(value);
    }

    static public explicit operator int(BinaryNumeral binary)
    {
        return (binary.value);
    }

    static public implicit operator string(BinaryNumeral binary)
    {
        return ("Conversion not yet implemented");
    }
}

class TestConversions
{
    static void Main()
    {
```

```
RomanNumeral roman;
BinaryNumeral binary;

roman = 10;

// Perform a conversion from a RomanNumeral to a BinaryNumeral:
binary = (BinaryNumeral)(int)roman;

// Perform a conversion from a BinaryNumeral to a RomanNumeral:
// No cast is required:
roman = binary;

System.Console.WriteLine((int)binary);
System.Console.WriteLine(binary);
}

}
```

输出

```
10
Conversion not yet implemented
```

可靠编程

- 在上面的示例中，语句：

C#

```
binary = (BinaryNumeral)(int)roman;
```

执行从 RomanNumeral 到 BinaryNumeral 的转换。由于没有从 RomanNumeral 到 BinaryNumeral 的直接转换，所以使用一个转换将 RomanNumeral 转换为 int，并使用另一个转换将 int 转换为 BinaryNumeral。

- 另外，语句

C#

```
roman = binary;
```

执行从 BinaryNumeral 到 RomanNumeral 的转换。由于 RomanNumeral 定义了从 BinaryNumeral 的隐式转换，所以不需要转换。

请参见

概念

[C# 编程指南](#)

[转换运算符 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

如何：使用运算符重载创建复数类(C# 编程指南)

本示例展示如何使用运算符重载定义复数加法的复数类 Complex。本程序使用 **ToString** 方法的重载显示数字的虚部和实部以及加法结果。

示例

C#

```

public struct Complex
{
    public int real;
    public int imaginary;

    public Complex(int real, int imaginary) //constructor
    {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Declare which operator to overload (+),
    // the types that can be added (two Complex objects),
    // and the return type (Complex):
    public static Complex operator +(Complex c1, Complex c2)
    {
        return new Complex(c1.real + c2.real, c1.imaginary + c2.imaginary);
    }

    // Override the ToString() method to display a complex number in the traditional format
    :
    public override string ToString()
    {
        return (System.String.Format("{0} + {1}i", real, imaginary));
    }
}

class TestComplex
{
    static void Main()
    {
        Complex num1 = new Complex(2, 3);
        Complex num2 = new Complex(3, 4);

        // Add two Complex objects through the overloaded plus operator:
        Complex sum = num1 + num2;

        // Print the numbers and the sum using the overridden ToString method:
        System.Console.WriteLine("First complex number: {0}", num1);
        System.Console.WriteLine("Second complex number: {0}", num2);
        System.Console.WriteLine("The sum of the two numbers: {0}", sum);
    }
}

```

输出

```

First complex number: 2 + 3i
Second complex number: 3 + 4i
The sum of the two numbers: 5 + 7i

```

请参见

参考

[C# 运算符](#)

[operator\(C# 参考\)](#)

概念

[C# 编程指南](#)

Equals() 和运算符 == 的重载准则(C# 编程指南)

C# 中有两种不同的相等：引用相等和值相等。值相等是大家普遍理解的意义上的相等：它意味着两个对象包含相同的值。例如，两个值为 2 的整数具有值相等性。引用相等意味着要比较的不是两个对象，而是两个对象引用，这两个对象引用引用的是同一个对象。这可以通过简单的赋值来实现，如下面的示例所示：

C#

```
System.Object a = new System.Object();
System.Object b = a;
System.Object.ReferenceEquals(a, b); //returns true
```

在上面的代码中，只存在一个对象，但存在对该对象的多个引用：a 和 b。由于它们引用的是同一个对象，因此具有引用相等性。如果两个对象具有引用相等性，则它们也具有值相等性，但是值相等性不能保证引用相等性。

若要检查引用相等性，应使用 [ReferenceEquals](#)。若要检查值相等性，应使用 [Equals](#) 或 [Equal](#)。

重写 Equals

Equals 是一个虚方法，允许任何类重写其实现。表示某个值（本质上可以是任何值类型）或一组值（如复数类）的任何类都应该重写 **Equals**。如果类型要实现 [IComparable](#)，则它应该重写 **Equals**。

Equals 的新实现应该遵循 [Equals](#) 的所有保证：

- x.Equals(x) 返回 true。
- x.Equals(y) 与 y.Equals(x) 返回相同的值。
- 如果 (x.Equals(y) && y.Equals(z)) 返回 true，则 x.Equals(z) 返回 true。
- 只要不修改 x 和 y 所引用的对象，x.Equals(y) 的后续调用就返回相同的值。
- x.Equals(null) 返回 false。

Equals 的新实现不应该引发异常。建议重写 **Equals** 的任何类同时也重写 [System.Object.GetHashCode](#)。除了实现 **Equals**（对象）外，还建议所有的类为自己的类型实现 **Equals**（类型）以增强性能。例如：

C#

```
class TwoDPoint : System.Object
{
    public readonly int x, y;

    public TwoDPoint(int x, int y) //constructor
    {
        this.x = x;
        this.y = y;
    }

    public override bool Equals(System.Object obj)
    {
        // If parameter is null return false.
        if (obj == null)
        {
            return false;
        }

        // If parameter cannot be cast to Point return false.
        TwoDPoint p = obj as TwoDPoint;
        if ((System.Object)p == null)
        {
            return false;
        }

        // Return true if the fields match:
        return (x == p.x) && (y == p.y);
    }
}
```

```

public bool Equals(TwoDPoint p)
{
    // If parameter is null return false:
    if ((object)p == null)
    {
        return false;
    }

    // Return true if the fields match:
    return (x == p.x) && (y == p.y);
}

public override int GetHashCode()
{
    return x ^ y;
}
}

```

可调用基类的 **Equals** 的任何派生类在完成其比较之前都应该这样做。在下面的示例中，**Equals** 调用基类 **Equals**，后者将检查空参数并将参数的类型与派生类的类型做比较。这样就把检查派生类中声明的新数据字段的任务留给了派生类中的 **Equals** 实现：

C#

```

class ThreeDPoint : TwoDPoint
{
    public readonly int z;

    public ThreeDPoint(int x, int y, int z)
        : base(x, y)
    {
        this.z = z;
    }

    public override bool Equals(System.Object obj)
    {
        // If parameter cannot be cast to ThreeDPoint return false:
        ThreeDPoint p = obj as ThreeDPoint;
        if ((object)p == null)
        {
            return false;
        }

        // Return true if the fields match:
        return base.Equals(obj) && z == p.z;
    }

    public bool Equals(ThreeDPoint p)
    {
        // Return true if the fields match:
        return base.Equals((TwoDPoint)p) && z == p.z;
    }

    public override int GetHashCode()
    {
        return base.GetHashCode() ^ z;
    }
}

```

Overriding Operator ==

默认情况下，运算符 `==` 通过判断两个引用是否指示同一对象来测试引用是否相等，因此引用类型不需要实现运算符 `==` 就能获得此功能。当类型不可变时，意味着实例中包含的数据不可更改，此时通过重载运算符 `==` 来比较值是否相等而不是比较引用是否相等可能会很有用，因为作为不可变的对象，只要它们具有相同的值，就可以将它们看作是相同的。建议不要在非不可变

类型中重写运算符 `==`。

重载的运算符 `==` 实现不应引发异常。重载运算符 `==` 的任何类型还应重载运算符 `!=`。例如：

C#

```
//add this code to class ThreeDPoint as defined previously
//
public static bool operator ==(ThreeDPoint a, ThreeDPoint b)
{
    // If both are null, or both are same instance, return true.
    if (System.Object.ReferenceEquals(a, b))
    {
        return true;
    }

    // If one is null, but not both, return false.
    if (((object)a == null) || ((object)b == null))
    {
        return false;
    }

    // Return true if the fields match:
    return a.x == b.x && a.y == b.y && a.z == b.z;
}

public static bool operator !=(ThreeDPoint a, ThreeDPoint b)
{
    return !(a == b);
}
```

注意

运算符 `==` 的重载中的常见错误是使用 `(a == b)`、`(a == null)` 或 `(b == null)` 来检查引用相等性。这会导致调用重载的运算符 `==`，从而导致无限循环。应使用 **ReferenceEquals** 或将类型强制转换为 **Object** 来避免无限循环。

请参见

参考

[语句、表达式和运算符\(C# 编程指南\)](#)

[运算符\(C# 编程指南\)](#)

[可重载运算符\(C# 编程指南\)](#)

[== 运算符\(C# 参考\)](#)

[\(\) 运算符\(C# 参考\)](#)

概念

[C# 编程指南](#)

对象、类和结构(C# 编程指南)

C# 是面向对象的编程语言，它使用类和结构来实现类型(如 Windows 窗体、用户界面控件和数据结构等)。典型的 C# 应用程序由程序员定义的类和 .NET Framework 的类组成。

C# 提供了许多功能强大的类定义方式，例如，提供不同的访问级别，从其他类继承功能，允许程序员指定实例化或销毁类型时的操作。

还可以通过使用类型参数将类定义为泛型，通过类型参数，客户端代码可以类型安全的有效方式来自定义类。客户端代码可以使用单个泛型类(例如，.NET Framework 类库中的 `System.Collections.Generic.List`)存储整数、字符串或任何其他类型的对象。

概述

对象、类和结构具有以下特点：

- 对象是给定数据类型的实例。在执行应用程序时，数据类型为创建(或实例化)的对象提供蓝图。
- 新数据类型是使用类和结构定义的。
- 类和结构(包含代码和数据)组成了 C# 应用程序的生成块。C# 应用程序始终包含至少一个类。
- 结构可视为轻量类，是创建用于存储少量数据的数据类型的理想选择，不能表示以后可能要通过继承进行扩展的类型。
- C# 类支持继承，这意味着它们可以从先前定义的类中派生。

相关章节

- [类型设计准则](#)
- [对象](#)
- [类](#)
- [结构](#)
- [继承](#)
- [成员](#)
- [方法](#)
- [构造函数](#)
- [析构函数](#)
- [使用构造函数和析构函数](#)
- [字段](#)
- [常数](#)
- [嵌套类型](#)
- [访问修饰符](#)
- [分部类定义](#)
- [静态类和静态类成员](#)
- [如何：了解传递结构与传递对方法的类引用两者之间的区别](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6 类和对象
- 1.7 结构
- 3.4.4 类成员
- 4.2.1 类类型

- 10 类
- 11 结构

请参见

概念

[C# 编程指南](#)

对象(C# 编程指南)

对象是具有数据、行为和标识的编程结构。对象数据包含在对象的字段、属性和事件中，对象行为则由对象的方法和接口定义。

对象具有标识 -- 数据集相同的两个对象不一定是同一对象。

C# 中的对象通过 **classes** 和 **structs** 定义 -- 该类型的所有对象都按照它们构成的同一蓝图操作。

对象概述

对象具有以下特点：

- C# 中使用的全都是对象，包括 Windows 窗体和控件。
- 对象是实例化的；也就是说，对象是从类和结构所定义的模板中创建的。
- 对象使用[属性\(C# 编程指南\)](#)获取和更改它们所包含的信息。
- 对象通常具有允许它们执行操作的方法和事件。
- Visual Studio 提供了操作对象的工具：使用[“属性”窗口](#)可以更改对象（如 Windows 窗体）的属性。使用[对象浏览器](#)可以检查对象的内容。
- 所有 C# 对象都继承自 [Object](#)。

相关章节

更多信息：

- [类\(C# 编程指南\)](#)
- [结构\(C# 编程指南\)](#)
- [构造函数\(C# 编程指南\)](#)
- [析构函数\(C# 编程指南\)](#)
- [事件\(C# 编程指南\)](#)

请参见

参考

[object\(C# 参考\)](#)

[继承\(C# 编程指南\)](#)

[class\(C# 参考\)](#)

[struct\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[远程对象](#)

类(C# 编程指南)

类是 C# 中功能最为强大的数据类型。像结构一样，类也定义了数据类型的数据和行为。然后，程序员可以创建作为此类的实例的对象。与结构不同，类支持继承，而继承是面向对象编程的基础部分。有关更多信息，请参见[继承](#)。

声明类

类是使用 `class` 关键字来定义的，如下面的示例所示：

C#

```
public class Customer
{
    //Fields, properties, methods and events go here...
}
```

`class` 关键字前面是访问级别。在该例中，使用了 `public`，这表示任何人都可以基于该类创建对象。类的名称位于 `class` 关键字的后面。定义的其余部分是类的主体，用于定义行为和数据。类的字段、属性、方法和事件统称为“类成员”。

创建对象

尽管有时类和对象可互换，但它们是不同的概念。类定义对象的类型，但它不是对象本身。对象是基于类的具体实体，有时称为类的实例。

通过使用 `new` 关键字，后跟对象将基于的类的名称，可以创建对象，如下所示：

C#

```
Customer object1 = new Customer();
```

创建类的实例后，将向程序员传递回对该对象的引用。在上例中，`object1` 是对基于 `Customer` 的对象的引用。此引用引用新对象，但不包含对象数据本身。实际上，可以在根本不创建对象的情况下创建对象引用：

C#

```
Customer object2;
```

建议不要创建像这样的不引用对象的对象引用，因为在运行时通过这样的引用来访问对象的尝试将会失败。但是，可以创建这样的引用来引用对象，方法是创建新对象，或者将它分配给现有的对象，如下所示：

C#

```
Customer object3 = new Customer();
Customer object4 = object3;
```

此代码创建了两个对象引用，它们引用同一个对象。因此，通过 `object3` 对对象所做的任何更改都将反映在随后使用的 `object4` 中。这是因为基于类的对象是按引用来引用的，因此类称为引用类型。

类继承

继承是通过使用派生来实现的，而派生意味着类是使用基类声明的，它的数据和行为从基类继承。通过在派生的类名后面追加冒号和基类名称，可以指定基类，如下所示：

C#

```
public class Manager : Employee
{
    // Employee fields, properties, methods and events are inherited
    // New Manager fields, properties, methods and events go here...
}
```

当类声明基类时，为基类定义的所有类成员也成为新类的一部分。因为基类自身可能继承自另一个类，而后者又从另一个类继承，依此类推，最终类可能具有很多个基类。

示例

下面的示例定义一个公共类，其中包含一个字段、一个方法和一个称为构造函数的特殊方法。有关更多信息，请参见[构造函数](#)。然后使用 `new` 关键字将该类实例化。

C#

```
public class Person
{
    // Field
    public string name;

    // Constructor
    public Person()
    {
        name = "unknown";
    }

    // Method
    public void SetName(string newName)
    {
        name = newName;
    }
}
class TestPerson
{
    static void Main()
    {
        Person person1 = new Person();
        System.Console.WriteLine(person1.name);

        person1.SetName("John Smith");
        System.Console.WriteLine(person1.name);
    }
}
```

输出

```
unknown
John Smith
```

类概述

类具有以下特点：

- 与 C++ 不同，C# 只支持单继承：类只能从一个基类继承实现。
- 一个类可以实现多个接口。有关更多信息，请参见[接口](#)。
- 类定义可在不同的源文件之间进行拆分。有关更多信息，请参见[分部类定义](#)。
- 静态类是仅包含静态方法的密封类。有关更多信息，请参见[静态类和静态类成员](#)。
-

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.6 类和对象
- 10 类

请参见 参考

[成员 \(C# 编程指南\)](#)
[方法 \(C# 编程指南\)](#)
[析构函数 \(C# 编程指南\)](#)
[对象 \(C# 编程指南\)](#)
概念
[C# 编程指南](#)
[其他资源](#)
[构造函数 \(C# 编程指南\)](#)

结构(C# 编程指南)

结构是使用 `struct` 关键字定义的，例如：

C#

```
public struct PostalAddress
{
    // Fields, properties, methods and events go here...
}
```

结构与类共享几乎所有相同的语法，但结构比类受到的限制更多：

- 在结构声明中，除非字段被声明为 `const` 或 `static`，否则无法初始化。
- 结构不能声明默认构造函数(没有参数的构造函数)或析构函数。

结构的副本由编译器自动创建和销毁，因此不需要使用默认构造函数和析构函数。实际上，编译器通过为所有字段赋予默认值(参见[默认值表](#))来实现默认构造函数。结构不能从类或其他结构继承。

结构是值类型 -- 如果从结构创建一个对象并将该对象赋给某个变量，变量则包含结构的全部值。复制包含结构的变量时，将复制所有数据，对新副本所做的任何修改都不会改变旧副本的数据。由于结构不使用引用，因此结构没有标识 -- 具有相同数据的两个值类型实例是无法区分的。C# 中的所有值类型本质上都继承自 `ValueType`，后者继承自 `Object`。

编译器可以在一个称为装箱的过程中将值类型转换为引用类型。有关更多信息，请参见[装箱和取消装箱](#)。

结构概述

结构具有以下特点：

- 结构是值类型，而类是引用类型。
- 与类不同，结构的实例化可以不使用 `new` 运算符。
- 结构可以声明构造函数，但它们必须带参数。
- 一个结构不能从另一个结构或类继承，而且不能作为一个类的基。所有结构都直接继承自 `System.ValueType`，后者继承自 `System.Object`。
- 结构可以实现接口。

相关章节

更多信息：

- [使用结构](#)
- [使用构造函数和析构函数](#)
- [如何：了解传递结构与传递对方法的类引用两者之间的区别](#)
- [如何：在结构之间实现用户定义的转换](#)

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

结构设计

[类\(C# 编程指南\)](#)

使用结构(C# 编程指南)

struct 类型适于表示 **Point**、**Rectangle** 和 **Color** 等轻量对象。尽管可以将一个点表示为类，但在某些情况下，使用结构更有效。例如，如果声明一个 1000 个 **Point** 对象组成的数组，为了引用每个对象，则需分配更多内存；这种情况下，使用结构可以节约资源。由于 .NET Framework 包含名为 **Point** 的对象，因此我们转而调用结构“CoOrds”。

C#

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

声明结构的默认(无参数)构造函数是错误的。总是提供默认构造函数以将结构成员初始化为它们的默认值。在结构中初始化实例字段也是错误的。

如果使用 **new** 运算符创建结构对象，则会创建该结构对象，并调用适当的构造函数。与类不同，结构的实例化可以不使用 **new** 运算符。如果不使用 **new**，则在初始化所有字段之前，字段都保持未赋值状态且对象不可用。

对于结构，不像类那样存在继承。一个结构不能从另一个结构或类继承，而且不能作为一个类的基。但是，结构从基类 **Object** 继承。结构可实现接口，其方式同类完全一样。

与 C++ 不同，无法使用 **struct** 关键字声明类。在 C# 中，类与结构在语义上是不同的。结构是值类型，而类是引用类型。有关更多信息，请参见[值类型](#)。

除非需要引用类型语义，否则系统将较小的类作为结构处理效率会更高。

示例 1

下面的示例演示使用默认构造函数和参数化构造函数的 **struct** 初始化。

C#

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

C#

```
// Declare and initialize struct objects.
class TestCoOrds
{
    static void Main()
    {
        // Initialize:
        CoOrds coords1 = new CoOrds();
        CoOrds coords2 = new CoOrds(10, 10);

        // Display results:
        System.Console.Write("CoOrds 1: ");
        System.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);
    }
}
```

```
        System.Console.Write("CoOrds 2: ");
        System.Console.WriteLine("x = {0}, y = {1}", coords2.x, coords2.y);
    }
}
```

输出

```
CoOrds 1: x = 0, y = 0
CoOrds 2: x = 10, y = 10
```

示例 2

下面举例说明了结构特有的一种功能。它在不使用 **new** 运算符的情况下创建 CoOrds 对象。如果将 **struct** 换成 **class**，程序将不会编译。

C#

```
public struct CoOrds
{
    public int x, y;

    public CoOrds(int p1, int p2)
    {
        x = p1;
        y = p2;
    }
}
```

C#

```
// Declare a struct object without "new."
class TestCoOrdsNoNew
{
    static void Main()
    {
        // Declare an object:
        CoOrds coords1;

        // Initialize:
        coords1.x = 10;
        coords1.y = 20;

        // Display results:
        System.Console.Write("CoOrds 1: ");
        System.Console.WriteLine("x = {0}, y = {1}", coords1.x, coords1.y);
    }
}
```

输出

```
CoOrds 1: x = 10, y = 20
```

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[结构\(C# 编程指南\)](#)

继承(C# 编程指南)

类可以从其他类中继承。这是通过以下方式实现的：在声明类时，在类名称后放置一个冒号，然后在冒号后指定要从中继承的类（即基类）。例如：

C#

```
public class A
{
    public A() { }
}

public class B : A
{
    public B() { }
}
```

新类（即派生类）将获取基类的所有非私有数据和行为以及新类为自己定义的所有其他数据或行为。因此，新类具有两个有效类型：新类的类型和它继承的类的类型。

在上面的示例中，类 B 既是有效的 B，又是有效的 A。访问 B 对象时，可以使用强制转换操作将其转换为 A 对象。强制转换不会更改 B 对象，但您的 B 对象视图将限制为 A 的数据和行为。将 B 强制转换为 A 后，可以将该 A 重新强制转换为 B。并非 A 的所有实例都可强制转换为 B，只有实际上是 B 的实例的那些实例才可以强制转换为 B。如果将类 B 作为 B 类型访问，则可以同时获得类 A 和类 B 的数据和行为。对象可以表示多个类型的能力称为多态性。有关更多信息，请参见[多态性\(C# 编程指南\)](#)。有关强制转换的更多信息，请参见[强制转换\(C# 编程指南\)](#)。

结构不能从其他结构或类中继承。类和结构都可以从一个或多个接口中继承。有关更多信息，请参见[接口\(C# 编程指南\)](#)。

本节内容

- [抽象类、密封类及类成员\(C# 编程指南\)](#)
- [多态性\(C# 编程指南\)](#)
- [接口\(C# 编程指南\)](#)

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[class\(C# 参考\)](#)

[struct\(C# 参考\)](#)

概念

[C# 编程指南](#)

抽象类、密封类及类成员 (C# 编程指南)

使用 `abstract` 关键字可以创建仅用于继承用途的类和类成员，即定义派生的非抽象类的功能。使用 `sealed` 关键字可以防止继承以前标记为 `virtual` 的类或某些类成员。有关更多信息，请参见[如何: 定义抽象属性 \(C# 编程指南\)](#)。

抽象类和类成员

可以将类声明为抽象类。方法是在类定义中将关键字 `abstract` 置于关键字 `class` 的前面。例如：

C#

```
public abstract class A
{
    // Class members here.
}
```

抽象类不能实例化。抽象类的用途是提供多个派生类可共享的基类的公共定义。例如，类库可以定义一个作为其多个函数的参数的抽象类，并要求程序员使用该库通过创建派生类来提供自己的类实现。

抽象类也可以定义抽象方法。方法是将关键字 `abstract` 添加到方法的返回类型的前面。例如：

C#

```
public abstract class A
{
    public abstract void DoWork(int i);
}
```

抽象方法没有实现，所以方法定义后面是分号，而不是常规的方法块。抽象类的派生类必须实现所有抽象方法。当抽象类从基类继承虚方法时，抽象类可以使用抽象方法重写该虚方法。例如：

C#

```
// compile with: /target:library
public class D
{
    public virtual void DoWork(int i)
    {
        // Original implementation.
    }
}

public abstract class E : D
{
    public abstract override void DoWork(int i);
}

public class F : E
{
    public override void DoWork(int i)
    {
        // New implementation.
    }
}
```

如果将虚方法声明为抽象方法，则它对于从抽象类继承的所有类而言仍然是虚的。继承抽象方法的类无法访问该方法的原始实现。在前面的示例中，类 F 上的 `DoWork` 无法调用类 D 上的 `DoWork`。在此情况下，抽象类可以强制派生类为虚方法提供新的方法实现。

密封类和类成员

可以将类声明为密封类。方法是在类定义中将关键字 `sealed` 置于关键字 `class` 的前面。例如：

C#

```
public sealed class D
{
    // Class members here.
}
```

密封类不能用作基类。因此，它也不能是抽象类。密封类主要用于防止派生。由于密封类从不用作基类，所以有些运行时优化可以使对密封类成员的调用略快。

在对基类的虚成员进行重写的派生类上的类成员、方法、字段、属性或事件可以将该成员声明为密封成员。在用于以后的派生类时，这将取消成员的虚效果。方法是在类成员声明中将 **sealed** 关键字置于 **override** 关键字的前面。例如：

C#

```
public class D : C
{
    public sealed override void DoWork() { }
```

请参见

任务

[如何：定义抽象属性\(C# 编程指南\)](#)

参考

[对象、类和结构\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

[方法\(C# 编程指南\)](#)

[字段\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

如何：定义抽象属性 (C# 编程指南)

下面的示例演示如何定义 **抽象属性**。抽象属性声明不提供属性访问器的实现，它只声明该类支持属性，而将访问器实现留给派生类。下面的示例演示如何实现从基类继承的抽象属性。

此示例由三个文件组成，其中每个文件都单独编译，产生的程序集由下一次编译引用：

- `abstractshape.cs`: 包含抽象 `Area` 属性的 `Shape` 类。
- `shapes.cs`: `Shape` 类的子类。
- `shapetest.cs`: 测试程序，它显示某些 `Shape` 派生对象的面积。

若要编译该示例，请使用以下命令：

```
csc abstractshape.cs shapes.cs shapetest.cs
```

这样将生成可执行文件 `shapetest.exe`。

示例

该文件声明的 `Shape` 类包含 **double** 类型的 `Area` 属性。

C#

```
// compile with: csc /target:library abstractshape.cs
public abstract class Shape
{
    private string m_id;

    public Shape(string s)
    {
        // calling the set accessor of the Id property.
        Id = s;
    }

    public string Id
    {
        get
        {
            return m_id;
        }

        set
        {
            m_id = value;
        }
    }

    // Area is a read-only property - only a get accessor is needed:
    public abstract double Area
    {
        get;
    }

    public override string ToString()
    {
        return Id + " Area = " + string.Format("{0:F2}", Area);
    }
}
```

- 属性的修饰符就放置在属性声明中。例如：

```
public abstract double Area
```

- 声明抽象属性时(如本示例中的 Area)，指明哪些属性访问器可用即可，不要实现它们。在此示例中，只有一个 get 访问器可用，因此该属性是只读的。

下面的代码演示 Shape 的三个子类，并演示它们如何重写 Area 属性来提供自己的实现。

C#

```
// compile with: csc /target:library /reference:abstractshape.dll shapes.cs
public class Square : Shape
{
    private int m_side;

    public Square(int side, string id)
        : base(id)
    {
        m_side = side;
    }

    public override double Area
    {
        get
        {
            // Given the side, return the area of a square:
            return m_side * m_side;
        }
    }
}

public class Circle : Shape
{
    private int m_radius;

    public Circle(int radius, string id)
        : base(id)
    {
        m_radius = radius;
    }

    public override double Area
    {
        get
        {
            // Given the radius, return the area of a circle:
            return m_radius * m_radius * System.Math.PI;
        }
    }
}

public class Rectangle : Shape
{
    private int m_width;
    private int m_height;

    public Rectangle(int width, int height, string id)
        : base(id)
    {
        m_width = width;
        m_height = height;
    }

    public override double Area
    {
        get
        {
            // Given the width and height, return the area of a rectangle:
            return m_width * m_height;
        }
    }
}
```

```
}
```

下面的代码演示一个测试程序，它创建若干 Shape 派生对象，并输出它们的面积。

C#

```
// compile with: csc /reference:abstractshape.dll;shapes.dll shapetest.cs
class TestClass
{
    static void Main()
    {
        Shape[] shapes =
        {
            new Square(5, "Square #1"),
            new Circle(3, "Circle #1"),
            new Rectangle( 4, 5, "Rectangle #1")
        };

        System.Console.WriteLine("Shapes Collection");
        foreach (Shape s in shapes)
        {
            System.Console.WriteLine(s);
        }
    }
}
```

输出

```
Shapes Collection
Square #1 Area = 25.00
Circle #1 Area = 28.27
Rectangle #1 Area = 20.00
```

[请参见](#)

[参考](#)

[对象、类和结构\(C# 编程指南\)](#)

[抽象类、密封类及类成员\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

多态性(C# 编程指南)

通过继承，一个类可以用作多种类型：可以用作它自己的类型、任何基类型，或者在实现接口时用作任何接口类型。这称为多态性。C# 中的每种类型都是多态的。类型可用作它们自己的类型或用作 Object 实例，因为任何类型都自动将 Object 当作基类型。

多态性不仅对派生类很重要，对基类也很重要。任何情况下，使用基类实际上都可能是在使用已强制转换为基类类型的派生类对象。基类的设计者可以预测到其基类中可能会在派生类中发生更改的方面。例如，表示汽车的基类可能包含这样的行为：当考虑的汽车为小型货车或敞篷汽车时，这些行为将会改变。基类可以将这些类成员标记为虚拟的，从而允许表示敞篷汽车和小型货车的派生类重写该行为。

有关更多信息，请参见[继承](#)。

多态性概述

当派生类从基类继承时，它会获得基类的所有方法、字段、属性和事件。若要更改基类的数据和行为，您有两种选择：可以使用新的派生成员替换基成员，或者可以重写虚拟的基成员。

使用新的派生成员替换基类的成员需要使用 new 关键字。如果基类定义了一个方法、字段或属性，则 new 关键字用于在派生类中创建该方法、字段或属性的新定义。new 关键字放置在要替换的类成员的返回类型之前。例如：

C#

```
public class BaseClass
{
    public void DoWork() { }
    public int WorkField;
    public int WorkProperty
    {
        get { return 0; }
    }
}

public class DerivedClass : BaseClass
{
    public new void DoWork() { }
    public new int WorkField;
    public new int WorkProperty
    {
        get { return 0; }
    }
}
```

使用 new 关键字时，调用的是新的类成员而不是已被替换的基类成员。这些基类成员称为隐藏成员。如果将派生类的实例强制转换为基类的实例，就仍然可以调用隐藏类成员。例如：

C#

```
DerivedClass B = new DerivedClass();
B.DoWork(); // Calls the new method.

BaseClass A = (BaseClass)B;
A.DoWork(); // Calls the old method.
```

为了使派生类的实例完全接替来自基类的类成员，基类必须将该成员声明为虚拟的。这是通过在该成员的返回类型之前添加 virtual 关键字来实现的。然后，派生类可以选择使用 override 关键字而不是 new，将基类实现替换为它自己的实现。例如：

C#

```
public class BaseClass
{
    public virtual void DoWork() { }
    public virtual int WorkProperty
    {
```

```

        get { return 0; }
    }
}

public class DerivedClass : BaseClass
{
    public override void DoWork() { }

    public override int WorkProperty
    {
        get { return 0; }
    }
}

```

字段不能是虚拟的，只有方法、属性、事件和索引器才可以是虚拟的。当派生类重写某个虚拟成员时，即使该派生类的实例被当作基类的实例访问，也会调用该成员。例如：

C#

```

DerivedClass B = new DerivedClass();
B.DoWork(); // Calls the new method.

BaseClass A = (BaseClass)B;
A.DoWork(); // Also calls the new method.

```

使用虚拟方法和属性可以预先计划未来的扩展。由于在调用虚拟成员时不考虑调用方正在使用的类型，所以派生类可以选择完全更改基类的外观行为。

无论在派生类和最初声明虚拟成员的类之间已声明了多少个类，虚拟成员都将永远为虚拟成员。如果类 A 声明了一个虚拟成员，类 B 从 A 派生，类 C 从类 B 派生，则类 C 继承该虚拟成员，并且可以选择重写它，而不管类 B 是否为该成员声明了重写。例如：

C#

```

public class A
{
    public virtual void DoWork() { }
}

public class B : A
{
    public override void DoWork() { }
}

```

C#

```

public class C : B
{
    public override void DoWork() { }
}

```

派生类可以通过将重写声明为密封的来停止虚拟继承。这需要在类成员声明中将 `sealed` 关键字放在 `override` 关键字的前面。例如：

C#

```

public class C : B
{
    public sealed override void DoWork() { }
}

```

在上面的示例中，方法 `DoWork` 对从 C 派生的任何类都不再是虚拟的。它对 C 的实例仍然是虚拟的 -- 即使将这些实例强制转换为类型 B 或类型 A。派生类可以通过使用 `new` 关键字替换密封的方法，如下面的示例所示：

C#

```
public class D : C
{
    public new void DoWork() { }
```

在此情况下，如果在 D 中使用类型为 D 的变量调用 DoWork，被调用的将是新的 DoWork。如果使用类型为 C、B 或 A 的变量访问 D 的实例，对 DoWork 的调用将遵循虚拟继承的规则，即把这些调用传送到类 C 的 DoWork 实现。

已替换或重写某个方法或属性的派生类仍然可以使用基关键字访问基类的该方法或属性。例如：

C#

```
public class A
{
    public virtual void DoWork() { }

public class B : A
{
    public override void DoWork() { }
```

C#

```
public class C : B
{
    public override void DoWork()
    {
        // Call DoWork on B to get B's behavior:
        base.DoWork();

        // DoWork behavior specific to C goes here:
        // ...
    }
}
```

有关更多信息，请参见 [base](#)。

注意

建议虚拟成员在它们自己的实现中使用 **base** 来调用该成员的基类实现。允许基类行为发生使得派生类能够集中精力实现特定于派生类的行为。未调用基类实现时，由派生类负责使它们的行为与基类的行为兼容。

本节内容

更多信息：

- [使用 Override 和 New 关键字进行版本控制\(C# 编程指南\)](#)
- [了解何时使用 Override 和 New 关键字\(C# 编程指南\)](#)
- [如何：重写 ToString 方法\(C# 编程指南\)](#)

请参见

参考

[继承\(C# 编程指南\)](#)

[抽象类、密封类及类成员\(C# 编程指南\)](#)

[方法\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

[索引器\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[C# 编程指南](#)

使用 `Override` 和 `New` 关键字进行版本控制 (C# 编程指南)

C# 语言经过专门设计，以便不同库中的基类与派生类之间的版本控制可以不断向前发展，同时保持向后兼容。这具有多方面的意义。例如，这意味着在基类中引入与派生类中的某个成员具有相同名称的新成员在 C# 中是完全支持的，不会导致意外行为。它还意味着类必须显式声明某方法是要重写一个继承方法，还是一个仅隐藏具有类似名称的继承方法的新方法。

C# 允许派生类包含与基类方法名称相同的方法。

- 基类方法必须定义为 `virtual`。
- 如果派生类中的方法前面没有 `new` 或 `override` 关键字，则编译器将发出警告，该方法将有如存在 `new` 关键字一样执行操作。
- 如果派生类中的方法前面带有 `new` 关键字，则该方法被定义为独立于基类中的方法。
- 如果派生类中的方法前面带有 `override` 关键字，则派生类的对象将调用该方法，而不调用基类方法。
- 可以从派生类中使用 `base` 关键字调用基类方法。
- `override`、`virtual` 和 `new` 关键字还可以用于属性、索引器和事件中。

默认情况下，C# 方法不是虚方法 -- 如果将一种方法声明为虚方法，则继承该方法的任何类都可以实现其自己的版本。若要使方法成为虚方法，必须在基类的方法声明中使用 `virtual` 修饰符。然后，派生类可以使用 `override` 关键字重写基虚方法，或使用 `new` 关键字隐藏基类中的虚方法。如果 `override` 关键字和 `new` 关键字均未指定，编译器将发出警告，并且派生类中的方法将隐藏基类中的方法。有关更多信息，请参见[编译器警告 CS0108](#)。

为了在实践中演示上述情况，我们暂时假定公司 A 创建了一个名为 `GraphicsClass` 的类，您的程序使用该类。`GraphicsClass` 类似如下：

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
}
```

您的公司使用此类，并且您在添加新方法时将其用来派生自己的类：

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public void DrawRectangle() { }
```

您在应用程序的使用过程中没有遇到任何问题，直到公司 A 发布了 `GraphicsClass` 的新版本，该新版本类似如下：

C#

```
class GraphicsClass
{
    public virtual void DrawLine() { }
    public virtual void DrawPoint() { }
    public virtual void DrawRectangle() { }
}
```

现在，`GraphicsClass` 的新版本中包含了一个称为 `DrawRectangle` 的方法。最初，一切正常。新版本仍与旧版本二进制兼容 -- 即使在计算机系统中安装新类，部署的所有软件仍将继续工作。在您的派生类中，对方法 `DrawRectangle` 的任何现有调用将继续引用您的版本。

但是，一旦使用 `GraphicsClass` 的新版本重新编译应用程序，您将收到来自编译器的警告。有关更多信息，请参见[编译器警告 CS0108](#)。

此警告提示您需要考虑您的 DrawRectangle 方法在应用程序中的工作方式。

如果想用您的方法重写新的基类方法, 请使用 **override** 关键字, 如下所示:

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public override void DrawRectangle() { }
```

override 关键字可确保派生自 YourDerivedGraphicsClass 的任何对象都将使用 DrawRectangle 的派生类版本。派生自 YourDerivedGraphicsClass 的对象仍可以使用 **base** 关键字访问 DrawRectangle 的基类版本, 如下所示:

C#

```
base.DrawRectangle();
```

如果不使用您的方法重写新的基类方法, 则应注意下面的事项。为避免在两种方法之间引起混淆, 可以重命名您的方法。重命名方法可能很耗时且容易出错, 而且在某些情况下并不实用。但是, 如果您的项目相对较小, 则可以使用 Visual Studio 的重构选项来重命名方法。有关更多信息, 请参见[重构类和类型](#)。

或者, 也可以通过在派生类定义中使用关键字 **new** 来防止出现该警告, 如下所示:

C#

```
class YourDerivedGraphicsClass : GraphicsClass
{
    public new void DrawRectangle() { }
```

使用 **new** 关键字告诉编译器您的定义将隐藏基类中包含的定义。这是默认行为。

重写和方法选择

当在类中指定方法时, 如果有多个方法与调用兼容(例如, 存在两种同名的方法, 并且其参数与传递的参数兼容), 则 C# 编译器将选择最佳方法进行调用。下面的方法将是兼容的:

C#

```
public class Derived : Base
{
    public override void DoWork(int param) { }
    public void DoWork(double param) { }
}
```

在 Derived 的一个实例中调用 DoWork 时, C# 编译器将首先尝试使该调用与最初在 Derived 上声明的 DoWork 版本兼容。重写方法不被视为是在类上进行声明的, 而是在基类上声明的方法的新实现。仅当 C# 编译器无法将方法调用与 Derived 上的原始方法匹配时, 它才尝试将该调用与具有相同名称和兼容参数的重写方法匹配。例如:

C#

```
int val = 5;
Derived d = new Derived();
d.DoWork(val); // Calls DoWork(double).
```

由于变量 val 可以隐式转换为 double 类型, 因此 C# 编译器将调用 DoWork(double), 而不是 DoWork(int)。有两种方法可以避免此情况。首先, 避免将新方法声明为与虚方法同名。其次, 可以通过将 Derived 的实例强制转换为 Base 来使 C# 编译器搜索基类方法列表, 从而使其调用虚方法。由于是虚方法, 因此将调用 Derived 上的 DoWork(int) 的实现。例如:

C#

```
((Base)d).DoWork(val); // Calls DoWork(int) on Derived.
```

[请参见](#)

[参考](#)

[对象、类和结构\(C# 编程指南\)](#)

[方法\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

了解何时使用 `Override` 和 `New` 关键字 (C# 编程指南)

C# 允许派生类中的方法与基类中的方法具有相同的名称，只要您非常明确应如何处理新方法。下面的示例演示 `new` 和 `override` 关键字的使用。

首先声明三个类：一个名为 `Car` 的基类以及从该基类派生的两个类 `ConvertibleCar` 和 `Minivan`。基类包含一个可将有关汽车的描述发送到控制台的方法 (`DescribeCar`)。派生类方法也包含一个名为 `DescribeCar` 的方法，该方法显示派生类的独特属性。这些方法还调用基类的 `DescribeCar` 方法来演示从 `Car` 类继承属性的方式。

为了强调区别，`ConvertibleCar` 类使用 `new` 关键字来定义，而 `Minivan` 类使用 `override` 来定义。

C#

```
// Define the base class
class Car
{
    public virtual void DescribeCar()
    {
        System.Console.WriteLine("Four wheels and an engine.");
    }
}

// Define the derived classes
class ConvertibleCar : Car
{
    public new virtual void DescribeCar()
    {
        base.DescribeCar();
        System.Console.WriteLine("A roof that opens up.");
    }
}

class Minivan : Car
{
    public override void DescribeCar()
    {
        base.DescribeCar();
        System.Console.WriteLine("Carries seven people.");
    }
}
```

现在可以编写一些代码来声明这些类的实例，并调用它们的方法以便对象能够描述其自身：

C#

```
public static void TestCars1()
{
    Car car1 = new Car();
    car1.DescribeCar();
    System.Console.WriteLine("-----");

    ConvertibleCar car2 = new ConvertibleCar();
    car2.DescribeCar();
    System.Console.WriteLine("-----");

    Minivan car3 = new Minivan();
    car3.DescribeCar();
    System.Console.WriteLine("-----");
}
```

正如预期的那样，输出类似如下所示：

Four wheels and an engine.

Four wheels and an engine.

A roof that opens up.

Four wheels and an engine.

Carries seven people.

但是，在该代码接下来的一节中，我们声明了一个从 Car 基类派生的对象的数组。此数组能够存储 Car、ConvertibleCar 和 Minivan 对象。该数组的声明类似如下所示：

C#

```
public static void TestCars2()
{
    Car[] cars = new Car[3];
    cars[0] = new Car();
    cars[1] = new ConvertibleCar();
    cars[2] = new Minivan();
}
```

然后可以使用一个 **foreach** 循环来访问该数组中包含的每个 Car 对象，并调用 `DescribeCar` 方法，如下所示：

C#

```
foreach (Car vehicle in cars)
{
    System.Console.WriteLine("Car object: " + vehicle.GetType());
    vehicle.DescribeCar();
    System.Console.WriteLine("-----");
}
```

此循环的输出如下所示：

Car object: YourApplication.Car

Four wheels and an engine.

Car object: YourApplication.ConvertibleCar

Four wheels and an engine.

Car object: YourApplication.Minivan

Four wheels and an engine.

Carries seven people.

注意，ConvertibleCar 说明与您的预期不同。由于使用了 **new** 关键字来定义此方法，所调用的不是派生类方法，而是基类方法。Minivan 对象正确地调用重写方法，并产生预期的结果。

若要强制一个规则，要求从 Car 派生的所有类都必须实现 `DescribeCar` 方法，则应创建一个新的基类，将方法 `DescribeCar` 定义为 **abstract**。抽象方法不包含任何代码，仅包含方法签名。从此基类派生的任何类都必须提供 `DescribeCar` 的实现。有关更多信息，请参见[抽象](#)。

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[使用 Override 和 New 关键字进行版本控制\(C# 编程指南\)](#)

如何：重写 **ToString** 方法 (C# 编程指南)

C# 中的每个对象都继承 **ToString** 方法，此方法返回该对象的字符串表示形式。例如，所有 `int` 类型的变量都有一个 **ToString** 方法，从而允许将变量的内容作为字符串返回：

C#

```
int x = 42;
string strx = x.ToString();
System.Console.WriteLine(strx);
```

创建自定义类或结构时，应该重写 **ToString** 方法，以便向客户端代码提供类型信息。

安全注意

决定通过此方法提供什么信息时，应考虑该类或结构是否会被不受信任的代码使用。请务必确保您没有提供任何会被恶意代码利用的信息。

在类或结构中重写 **OnString** 方法：

1. 通过下面的修饰符和返回类型声明 **ToString** 方法：

```
public override string ToString(){}  
}
```

2. 实现该方法，使其返回一个字符串。

下面的示例不仅返回类的名称，还返回特定于该类的某个实例的数据。请注意，它还会在 `age` 变量上使用 **ToString** 方法，将 `int` 转换为可输出的字符串。

```
class Person
{
    string name;
    int age;
    SampleObject(string name, int age)
    {
        this.name = name;
        this.age = age;
    }
    public override string ToString()
    {
        string s = age.ToString();
        return "Person: " + name + " " + s;
    }
}
```

请参见

参考

[对象、类和结构 \(C# 编程指南\)](#)

[new \(C# 参考\)](#)

[override \(C# 参考\)](#)

[virtual \(C# 参考\)](#)

概念

[C# 编程指南](#)

接口 (C# 编程指南)

接口是使用 [interface](#) 关键字定义的。例如：

C#

```
interface IComparable
{
    int CompareTo(object obj);
}
```

接口描述可属于任何[类或结构](#)的一组相关行为。接口可由方法、属性、事件、索引器或这四种成员类型的任何组合构成。接口不能包含字段。接口成员一定是公共的。

类和结构可以按照类继承基类或结构的类似方式继承接口，但有两个例外：

- 类或结构可继承多个接口。
- 类或结构继承接口时，仅继承方法名称和签名，因为接口本身不包含实现。例如：

C#

```
public class Minivan : Car, IComparable
{
    public int CompareTo(object obj)
    {
        //implementation of CompareTo
        return 0; //if the Minivans are equal
    }
}
```

若要实现接口成员，类中的对应成员必须是公共的、非静态的，并且与接口成员具有相同的名称和签名。类的属性和索引器可以为接口上定义的属性或索引器定义额外的访问器。例如，接口可以声明一个带有 [get](#) 访问器的属性，而实现该接口的类可以声明同时带有 [get](#) 和 [set](#) 访问器的同一属性。但是，如果属性或索引器使用显式实现，则访问器必须匹配。

接口和接口成员是抽象的；接口不提供默认实现。有关更多信息，请参见[抽象类、密封类和类成员](#)。

[IComparable](#) 接口向对象的用户宣布该对象可以将自身与同一类型的其他对象进行比较，接口的用户不需要知道相关的实现方式。

接口可以继承其他接口。类可以通过其继承的基类或接口多次继承某个接口。在这种情况下，如果将该接口声明为新类的一部分，则类只能实现该接口一次。如果没有将继承的接口声明为新类的一部分，其实现将由声明它的基类提供。基类可以使用虚拟成员实现接口成员；在这种情况下，继承接口的类可通过重写虚拟成员来更改接口行为。有关虚拟成员的更多信息，请参见[多态性](#)。

接口概述

接口具有下列属性：

- 接口类似于抽象基类：继承接口的任何非抽象类型都必须实现接口的所有成员。
- 不能直接实例化接口。
- 接口可以包含事件、索引器、方法和属性。
- 接口不包含方法的实现。
- 类和结构可从多个接口继承。
- 接口自身可从多个接口继承。

本节内容

- [显式接口实现 \(C# 编程指南\)](#)

- [接口属性\(C# 编程指南\)](#)
- [接口中的索引器\(C# 编程指南\)](#)
- [如何: 实现接口事件\(C# 编程指南\)](#)
- [如何: 显式实现接口成员\(C# 编程指南\)](#)
- [如何: 使用继承显式实现接口成员\(C# 编程指南\)](#)

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

[方法\(C# 编程指南\)](#)

[抽象类、密封类及类成员\(C# 编程指南\)](#)

[方法\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

[索引器\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[多态性\(C# 编程指南\)](#)

[事件\(C# 编程指南\)](#)

显式接口实现(C# 编程指南)

如果类实现两个接口，并且这两个接口包含具有相同签名的成员，那么在类中实现该成员将导致两个接口都使用该成员作为它们的实现。例如：

C#

```
interface IControl
{
    void Paint();
}
interface ISurface
{
    void Paint();
}
class SampleClass : IControl, ISurface
{
    // Both ISurface.Paint and IControl.Paint call this method.
    public void Paint()
    {
    }
}
```

然而，如果两个接口成员执行不同的函数，那么这可能会导致其中一个接口的实现不正确或两个接口的实现都不正确。可以显式地实现接口成员 -- 即创建一个仅通过该接口调用并且特定于该接口的类成员。这是使用接口名称和一个句点命名该类成员来实现的。例如：

C#

```
public class SampleClass : IControl, ISurface
{
    void IControl.Paint()
    {
        System.Console.WriteLine("IControl.Paint");
    }
    void ISurface.Paint()
    {
        System.Console.WriteLine("ISurface.Paint");
    }
}
```

类成员 **IControl.Paint** 只能通过 **IControl** 接口使用，**ISurface.Paint** 只能通过 **ISurface** 使用。两个方法实现都是分离的，都不可以直接在类中使用。例如：

C#

```
SampleClass obj = new SampleClass();
//obj.Paint(); // Compiler error.

IControl c = (IControl)obj;
c.Paint(); // Calls IControl.Paint on SampleClass.

ISurface s = (ISurface)obj;
s.Paint(); // Calls ISurface.Paint on SampleClass.
```

显式实现还用于解决两个接口分别声明具有相同名称的不同成员(如属性和方法)的情况：

C#

```
interface ILeft
{
    int P { get; }
```

```
}
```

```
interface IRight
```

```
{
```

```
    int P();
```

```
}
```

为了同时实现两个接口，类必须对属性 P 和/或方法 P 使用显式实现以避免编译器错误。例如：

C#

```
class Middle : ILeft, IRight
```

```
{
```

```
    public int P() { return 0; }
```

```
    int ILeft.P { get { return 0; } }
```

```
}
```

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[接口\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

如何: 显式实现接口成员 (C# 编程指南)

本示例声明一个 [接口 IDimensions](#) 和一个类 Box, 该类显式实现接口成员 `getLength` 和 `getWidth`。通过接口实例 `dimensions` 访问这些成员。

示例

C#

```
interface IDimensions
{
    float getLength();
    float getWidth();
}

class Box : IDimensions
{
    float lengthInches;
    float widthInches;

    Box(float length, float width)
    {
        lengthInches = length;
        widthInches = width;
    }
    // Explicit interface member implementation:
    float IDimensions.getLength()
    {
        return lengthInches;
    }
    // Explicit interface member implementation:
    float IDimensions.getWidth()
    {
        return widthInches;
    }

    static void Main()
    {
        // Declare a class instance box1:
        Box box1 = new Box(30.0f, 20.0f);

        // Declare an interface instance dimensions:
        IDimensions dimensions = (IDimensions)box1;

        // The following commented lines would produce compilation
        // errors because they try to access an explicitly implemented
        // interface member from a class instance:
        //System.Console.WriteLine("Length: {0}", box1.getLength());
        //System.Console.WriteLine("Width: {0}", box1.getWidth());

        // Print out the dimensions of the box by calling the methods
        // from an instance of the interface:
        System.Console.WriteLine("Length: {0}", dimensions.getLength());
        System.Console.WriteLine("Width: {0}", dimensions.getWidth());
    }
}
```

输出

```
Length: 30
Width: 20
```

可靠编程

- 请注意 `Main` 方法中下列代码行被注释掉, 因为它们将产生编译错误。显式实现的接口成员不能从类实例访问:

C#

```
//System.Console.WriteLine("Length: {0}", box1.getLength());  
//System.Console.WriteLine("Width: {0}", box1.getWidth());
```

- 还请注意，**Main** 方法中的下列代码行成功输出框的尺寸，因为这些方法是从接口实例调用的：

C#

```
System.Console.WriteLine("Length: {0}", dimensions.getLength());  
System.Console.WriteLine("Width: {0}", dimensions.getWidth());
```

请参见

任务

[如何：使用继承显式实现接口成员 \(C# 编程指南\)](#)

参考

[对象、类和结构 \(C# 编程指南\)](#)

[接口 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

如何：使用继承显式实现接口成员 (C# 编程指南)

显式接口实现还允许程序员实现具有相同成员名称的两个接口，并为每个接口成员各提供一个实现。本示例同时以公制单位和英制单位显示框的尺寸。Box类实现 IEnglishDimensions 和 IMetricDimensions 两个接口，它们表示不同的度量系统。两个接口有相同的成员名 Length 和 Width。

示例

C#

```
// Declare the English units interface:  
interface IEnglishDimensions  
{  
    float Length();  
    float Width();  
}  
  
// Declare the metric units interface:  
interface IMetricDimensions  
{  
    float Length();  
    float Width();  
}  
  
// Declare the Box class that implements the two interfaces:  
// IEnglishDimensions and IMetricDimensions:  
class Box : IEnglishDimensions, IMetricDimensions  
{  
    float lengthInches;  
    float widthInches;  
  
    public Box(float length, float width)  
    {  
        lengthInches = length;  
        widthInches = width;  
    }  
  
    // Explicitly implement the members of IEnglishDimensions:  
    float IEnglishDimensions.Length()  
    {  
        return lengthInches;  
    }  
  
    float IEnglishDimensions.Width()  
    {  
        return widthInches;  
    }  
  
    // Explicitly implement the members of IMetricDimensions:  
    float IMetricDimensions.Length()  
    {  
        return lengthInches * 2.54f;  
    }  
  
    float IMetricDimensions.Width()  
    {  
        return widthInches * 2.54f;  
    }  
  
    static void Main()  
    {  
        // Declare a class instance box1:  
        Box box1 = new Box(30.0f, 20.0f);  
  
        // Declare an instance of the English units interface:  
        IEnglishDimensions eDimensions = (IEnglishDimensions)box1;
```

```

    // Declare an instance of the metric units interface:
    IMetricDimensions mDimensions = (IMetricDimensions)box1;

    // Print dimensions in English units:
    System.Console.WriteLine("Length(in): {0}", eDimensions.Length());
    System.Console.WriteLine("Width (in): {0}", eDimensions.Width());

    // Print dimensions in metric units:
    System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
    System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
}
}

```

输出

```

Length(in): 30
Width (in): 20
Length(cm): 76.2
Width (cm): 50.8

```

可靠编程

如果希望默认度量采用英制单位, 请正常实现 `Length` 和 `Width` 这两个方法, 并从 `IMetricDimensions` 接口显式实现 `Length` 和 `Width` 方法:

C#

```

// Normal implementation:
public float Length()
{
    return lengthInches;
}
public float Width()
{
    return widthInches;
}

// Explicit implementation:
float IMetricDimensions.Length()
{
    return lengthInches * 2.54f;
}
float IMetricDimensions.Width()
{
    return widthInches * 2.54f;
}

```

这种情况下, 可以从类实例访问英制单位, 而从接口实例访问公制单位:

C#

```

public static void Test()
{
    Box box1 = new Box(30.0f, 20.0f);
    IMetricDimensions mDimensions = (IMetricDimensions)box1;

    System.Console.WriteLine("Length(in): {0}", box1.Length());
    System.Console.WriteLine("Width (in): {0}", box1.Width());
    System.Console.WriteLine("Length(cm): {0}", mDimensions.Length());
    System.Console.WriteLine("Width (cm): {0}", mDimensions.Width());
}

```

请参见

任务

[如何: 显式实现接口成员 \(C# 编程指南\)](#)

[参考](#)

[对象、类和结构\(C# 编程指南\)](#)

[接口\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

成员 (C# 编程指南)

类和结构具有表示其数据和行为的成员。这些成员包括：

[字段 \(C# 编程指南\)](#)

字段是被视为类的一部分的对象实例，通常保存类数据。例如，日历类可能具有一个包含当前日期的字段。

[属性 \(C# 编程指南\)](#)

属性是类中可以像类中的字段一样访问的方法。属性可以为类字段提供保护，避免字段在对象不知道的情况下被更改。

[方法 \(C# 编程指南\)](#)

方法定义类可以执行的操作。方法可以接受提供输入数据的参数，并且可以通过参数返回输出数据。方法还可以不使用参数而直接返回值。

[事件 \(C# 编程指南\)](#)

事件是向其他对象提供有关事件发生(如单击按钮或成功完成某个方法)通知的一种方式。事件是使用委托来定义和触发的。有关更多信息，请参见[事件和委托](#)。

[运算符 \(C# 编程指南\)](#)

运算符是对操作数执行运算的术语或符号，如 +、*、< 等。可以重新定义运算符，以便可以对自定义数据类型执行运算。有关更多信息，请参见[可重载运算符 \(C# 编程指南\)](#)。

[索引器 \(C# 编程指南\)](#)

索引器允许以类似于数组的方式为对象建立索引。

[构造函数 \(C# 编程指南\)](#)

构造函数是在第一次创建对象时调用的方法。它们通常用于初始化对象的数据。

[析构函数 \(C# 编程指南\)](#)

析构函数是当对象即将从内存中移除时由运行库执行引擎调用的方法。它们通常用来确保需要释放的所有资源都得到了适当的处理。

[嵌套类型 \(C# 编程指南\)](#)

嵌套类型是在类或结构中声明的类型。嵌套类型通常用于描述仅由包含它们的类型所使用的对象。

请参见

参考

[方法 \(C# 编程指南\)](#)

[方法 \(C# 编程指南\)](#)

[析构函数 \(C# 编程指南\)](#)

[属性 \(C# 编程指南\)](#)

[字段 \(C# 编程指南\)](#)

[索引器 \(C# 编程指南\)](#)

[嵌套类型 \(C# 编程指南\)](#)

[运算符 \(C# 编程指南\)](#)

[可重载运算符 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[类 \(C# 编程指南\)](#)

[事件 \(C# 编程指南\)](#)

[事件和委托](#)

[其他资源](#)

[成员设计准则](#)

[构造函数 \(C# 编程指南\)](#)

方法(C# 编程指南)

"方法"是包含一系列语句的代码块。在 C# 中，每个执行指令都是在方法的上下文中完成的。

方法在类或结构中声明，声明时需要指定访问级别、返回值、方法名称以及任何方法参数。方法参数放在括号中，并用逗号隔开。空括号表示方法不需要参数。下面的类包含三个方法：

C#

```
class Motorcycle
{
    public void StartEngine() { }
    public void AddGas(int gallons) { }
    public int Drive(int miles, int speed) { return 0; }
}
```

调用对象的方法类似于访问字段。在对象名称之后，依次添加句点、方法名称和括号。参数在括号内列出，并用逗号隔开。因此，可以如下所示来调用 `Motorcycle` 类的方法：

C#

```
Motorcycle moto = new Motorcycle();

moto.StartEngine();
moto.AddGas(15);
moto.Drive(5, 20);
```

方法参数

如前面的代码段所示，如果要将参数传递给方法，只需在调用方法时在括号内提供这些参数即可。对于被调用的方法，传入的变量称为“参数”。

方法所接收的参数也是在一组括号中提供的，但必须指定每个参数的类型和名称。该名称不必与参数相同。例如：

C#

```
public static void PassesInteger()
{
    int fortyFour = 44;
    TakesInteger(fortyFour);
}
static void TakesInteger(int i)
{
    i = 33;
}
```

在这里，一个名为 `PassesInteger` 的方法向一个名为 `TakesInteger` 的方法传递参数。在 `PassesInteger` 内，该参数被命名为 `fortyFour`，但在 `TakesInteger` 中，它是名为 `i` 的参数。此参数只存在于 `TakesInteger` 方法内。其他任意多个变量都可以命名为 `i`，并且它们可以是任何类型，只要它们不是在此方法内部声明的参数或变量即可。

注意，`TakesInteger` 将新值赋给所提供的参数。有人可能认为一旦 `TakesInteger` 返回，此更改就会反映在 `PassesInteger` 方法中，但实际上变量 `fortyFour` 中的值将保持不变。这是因为 `int` 是“值类型”。默认情况下，将值类型传递给方法时，传递的是副本而不是对象本身。由于它们是副本，因此对参数所做的任何更改都不会在调用方法内部反映出来。之所以叫做值类型，是因为传递的是对象的副本而不是对象本身。传递的是值，而不是同一个对象。

有关传递值类型的更多信息，请参见[传递值类型参数\(C# 编程指南\)](#)。有关 C# 中的值类型的列表，请参见[值类型表\(C# 参考\)](#)。

这与“引用类型”不同，后者是按引用传递的。将基于引用类型的对象传递到方法时，不会创建对象的副本，而是创建并传递对用作方法参数的对象的引用。因此，通过此引用所进行的更改将反映在调用方法中。引用类型是使用 `class` 关键字创建的，如下所示：

C#

```
public class SampleRefType
{
    public int value;
}
```

现在，如果将基于此类型的对象传递给方法，它将按引用传递。例如：

C#

```
public static void TestRefType()
{
    SampleRefType rt = new SampleRefType();
    rt.value = 44;
    ModifyObject(rt);
    System.Console.WriteLine(rt.value);
}
static void ModifyObject(SampleRefType obj)
{
    obj.value = 33;
}
```

此示例的效果本质上与前一示例相同。但是，由于使用的是引用类型，因此 `ModifyObject` 所做的更改反映在 `TestRefType` 方法中创建的对象中。因此，`TestRefType` 方法将显示值 33。

有关更多信息，请参见[传递引用类型参数\(C# 编程指南\)](#)和[引用类型\(C# 参考\)](#)。

返回值

方法可以向调用方返回值。如果返回类型(方法名称前列出的类型)不是 `void`，则方法可以使用 `return` 关键字来返回值。如果语句中 `return` 关键字的后面是与返回类型匹配的值，则该语句将该值返回给方法调用方。`return` 关键字还会停止方法的执行。如果返回类型为 `void`，则可使用没有值的 `return` 语句来停止方法的执行。如果没有 `return` 关键字，方法执行到代码块末尾时即会停止。具有非 `void` 返回类型的方法才能使用 `return` 关键字返回值。例如，下面的两个方法使用 `return` 关键字来返回整数：

C#

```
class SimpleMath
{
    public int AddTwoNumbers(int number1, int number2)
    {
        return number1 + number2;
    }

    public int SquareANumber(int number)
    {
        return number * number;
    }
}
```

若要使用从方法返回的值，调用方法可以在本来使用同一类型的值就已足够的任何位置使用方法调用本身。还可以将返回值赋给变量：例如，下面的两个代码示例可实现相同的目的：

C#

```
int result = obj.AddTwoNumbers(1, 2);
obj.SquareANumber(result);
```

C#

```
obj.SquareANumber(obj.AddTwoNumbers(1, 2));
```

使用中间变量(本例中为 `result`)来存储值。这有助于增强代码的可读性, 如果要多次使用该值, 则可能必须使用中间变量。

有关更多信息, 请参见 [return\(C# 参考\)](#)。

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#)中的以下各章节:

- 1.6.5 方法

- 10.5 方法

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[访问修饰符\(C# 编程指南\)](#)

[静态类和静态类成员\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

[params\(C# 参考\)](#)

[return\(C# 参考\)](#)

[out\(C# 参考\)](#)

[ref\(C# 参考\)](#)

[传递参数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

传递参数(C# 编程指南)

在 C# 中，既可以通过值也可以通过引用传递参数。通过引用传递参数允许函数成员（方法、属性、索引器、运算符和构造函数）更改参数的值，并保持该更改。若要通过引用传递参数，请使用 **ref** 或 **out** 关键字。为简单起见，本主题的示例中只使用了 **ref** 关键字。有关 **ref** 和 **out** 之间的差异的更多信息，请参见 [ref、out 和使用 ref 和 out 传递数组](#)。例如：

C#

```
// Passing by value
static void Square(int x)
{
    // code...
}
```

C#

```
// Passing by reference
static void Square(ref int x)
{
    // code...
}
```

本主题包括下列章节：

- [传递值类型参数](#)
- [传递引用类型参数](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1.6.5.1 参数](#)
- [5.1.4 值参数](#)
- [5.1.5 引用参数](#)
- [5.1.6 输出参数](#)
- [10.5.1 方法参数](#)

请参见

参考

[方法\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

传递值类型参数(C# 编程指南)

值类型变量直接包含其数据，这与**引用类型**变量不同，后者包含对其数据的引用。因此，向方法传递值类型变量意味着向方法传递变量的一个副本。方法内发生的对参数的更改对该变量中存储的原始数据无任何影响。如果希望所调用的方法更改参数的值，必须使用 **ref** 或 **out** 关键字通过引用传递该参数。为了简单起见，下面的示例使用 **ref**。

示例：通过值传递值类型

下面的示例演示通过值传递值类型参数。通过值将变量 **n** 传递给方法 **SquareIt**。方法内发生的任何更改对变量的原始值无任何影响。

C#

```
class PassingValByVal
{
    static void SquareIt(int x)
        // The parameter x is passed by value.
        // Changes to x will not affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(n); // Passing the variable by value.
        System.Console.WriteLine("The value after calling the method: {0}", n);
    }
}
```

输出

```
The value before calling the method: 5
The value inside the method: 25
The value after calling the method: 5
```

代码讨论

变量 **n** 为值类型，包含其数据（值为 5）。当调用 **SquareIt** 时，**n** 的内容被复制到参数 **x** 中，在方法内将该参数求平方。但在 **Main** 中，**n** 的值在调用 **SquareIt** 方法前后是相同的。实际上，方法内发生的更改只影响局部变量 **x**。

示例：通过引用传递值类型

下面的示例除使用 **ref** 关键字传递参数以外，其余与上一示例相同。参数的值在调用方法后发生更改。

C#

```
class PassingValByRef
{
    static void SquareIt(ref int x)
        // The parameter x is passed by reference.
        // Changes to x will affect the original value of x.
    {
        x *= x;
        System.Console.WriteLine("The value inside the method: {0}", x);
    }
    static void Main()
    {
        int n = 5;
        System.Console.WriteLine("The value before calling the method: {0}", n);

        SquareIt(ref n); // Passing the variable by reference.
        System.Console.WriteLine("The value after calling the method: {0}", n);
    }
}
```

```
}
```

输出

```
The value before calling the method: 5
```

```
The value inside the method: 25
```

```
The value after calling the method: 25
```

代码讨论

本示例中，传递的不是 `n` 的值，而是对 `n` 的引用。参数 `x` 不是 `int` 类型，它是对 `int` 的引用（本例中为对 `n` 的引用）。因此，当在方法内对 `x` 求平方时，实际被求平方的是 `x` 所引用的项：`n`。

示例：交换值类型

更改所传递参数的值的常见示例是 `Swap` 方法，在该方法中传递 `x` 和 `y` 两个变量，然后使方法交换它们的内容。必须通过引用向 `Swap` 方法传递参数；否则，方法内所处理的将是参数的本地副本。以下是使用引用参数的 `Swap` 方法的示例：

```
C#
```

```
static void SwapByRef(ref int x, ref int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

调用该方法时，请在调用中使用 `ref` 关键字，如下所示：

```
C#
```

```
static void Main()
{
    int i = 2, j = 3;
    System.Console.WriteLine("i = {0} j = {1}" , i, j);

    SwapByRef (ref i, ref j);

    System.Console.WriteLine("i = {0} j = {1}" , i, j);
}
```

输出

```
i = 2 j = 3
```

```
i = 3 j = 2
```

请参见

参考

[传递参数\(C# 编程指南\)](#)

[传递引用类型参数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

传递引用类型参数(C# 编程指南)

引用类型的变量不直接包含其数据;它包含的是对其数据的引用。当通过值传递引用类型的参数时,有可能更改引用所指向的数据,如某类成员的值。但是无法更改引用本身的值;也就是说,不能使用相同的引用为新类分配内存并使之在块外保持。若要这样做,应使用 `ref` 或 `out` 关键字传递参数。为了简单起见,下面的示例使用 `ref`。

示例:通过值传递引用类型

下面的示例演示通过值向 `Change` 方法传递引用类型的参数 `arr`。由于该参数是对 `arr` 的引用,所以有可能更改数组元素的值。但是,试图将参数重新分配到不同的内存位置时,该操作仅在方法内有效,并不影响原始变量 `arr`。

C#

```
class PassingRefByVal
{
    static void Change(int[] pArray)
    {
        pArray[0] = 888; // This change affects the original element.
        pArray = new int[5] {-3, -1, -2, -3, -4}; // This change is local.
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }

    static void Main()
    {
        int[] arr = {1, 4, 5};
        System.Console.WriteLine("Inside Main, before calling the method, the first element
is: {0}", arr [0]);

        Change(arr);
        System.Console.WriteLine("Inside Main, after calling the method, the first element
is: {0}", arr [0]);
    }
}
```

输出

```
Inside Main, before calling the method, the first element is: 1
Inside the method, the first element is: -3
Inside Main, after calling the method, the first element is: 888
```

代码讨论

在上个示例中,数组 `arr` 为引用类型,在未使用 `ref` 参数的情况下传递给方法。在此情况下,将向方法传递指向 `arr` 的引用的一个副本。输出显示方法有可能更改数组元素的内容,在这种情况下,从 1 改为 888。但是,在 `Change` 方法内使用 `new` 运算符来分配新的内存部分,将使变量 `pArray` 引用新的数组。因此,这之后的任何更改都不会影响原始数组 `arr`(它是在 `Main` 内创建的)。实际上,本示例中创建了两个数组,一个在 `Main` 内,一个在 `Change` 方法内。

示例:通过引用传递引用类型

本示例除在方法头和调用中使用 `ref` 关键字以外,其余与上个示例相同。方法内发生的任何更改都会影响调用程序中的原始变量。

C#

```
class PassingRefByRef
{
    static void Change(ref int[] pArray)
    {
        // Both of the following changes will affect the original variables:
        pArray[0] = 888;
        pArray = new int[5] {-3, -1, -2, -3, -4};
        System.Console.WriteLine("Inside the method, the first element is: {0}", pArray[0]);
    }
}
```

```

static void Main()
{
    int[] arr = {1, 4, 5};
    System.Console.WriteLine("Inside Main, before calling the method, the first element
is: {0}", arr[0]);

    Change(ref arr);
    System.Console.WriteLine("Inside Main, after calling the method, the first element
is: {0}", arr[0]);
}

```

输出

Inside Main, before calling the method, the first element is: 1

Inside the method, the first element is: -3

Inside Main, after calling the method, the first element is: -3

代码讨论

方法内发生的所有更改都影响 `Main` 中的原始数组。实际上，使用 `new` 运算符对原始数组进行了重新分配。因此，调用 `Change` 方法后，对 `arr` 的任何引用都将指向 `Change` 方法中创建的五个元素的数组。

示例：交换两个字符串

交换字符串是通过引用传递引用类型参数的很好的示例。本示例中，`str1` 和 `str2` 两个字符串在 `Main` 中初始化，并作为由 `ref` 关键字修改的参数传递给 `SwapStrings` 方法。这两个字符串在该方法内以及 `Main` 内均进行交换。

C#

```

class SwappingStrings
{
    static void SwapStrings(ref string s1, ref string s2)
        // The string parameter is passed by reference.
        // Any changes on parameters will affect the original variables.
    {
        string temp = s1;
        s1 = s2;
        s2 = temp;
        System.Console.WriteLine("Inside the method: {0} {1}", s1, s2);
    }

    static void Main()
    {
        string str1 = "John";
        string str2 = "Smith";
        System.Console.WriteLine("Inside Main, before swapping: {0} {1}", str1, str2);

        SwapStrings(ref str1, ref str2);    // Passing strings by reference
        System.Console.WriteLine("Inside Main, after swapping: {0} {1}", str1, str2);
    }
}

```

输出

Inside Main, before swapping: John Smith

Inside the method: Smith John

Inside Main, after swapping: Smith John

代码讨论

本示例中，需要通过引用传递参数以影响调用程序中的变量。如果同时从方法头和方法调用中移除 `ref` 关键字，则调用程序中不会发生任何更改。

有关字符串的更多信息，请参见[字符串](#)。

[请参见](#)

[参考](#)

[传递参数\(C# 编程指南\)](#)

[使用 ref 和 out 传递数组\(C# 编程指南\)](#)

[ref\(C# 参考\)](#)

[引用类型\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

构造函数(C# 编程指南)

任何时候，只要创建[类或结构](#)，就会调用它的构造函数。类或结构可能有多个接受不同参数的构造函数。构造函数使得程序员可设置默认值、限制实例化以及编写灵活且便于阅读的代码。

如果您没有为对象提供构造函数，则默认情况下 C# 将创建一个构造函数，该构造函数实例化对象，并将所有成员变量设置为此处列出的默认值：[默认值表\(C# 参考\)](#)。静态类和结构也可以有构造函数。

本节内容

[使用构造函数\(C# 编程指南\)](#)

[实例构造函数\(C# 编程指南\)](#)

[私有构造函数\(C# 编程指南\)](#)

[静态构造函数\(C# 编程指南\)](#)

[如何：编写复制构造函数\(C# 编程指南\)](#)

相关章节

[C# 编程指南](#)

[对象、类和结构\(C# 编程指南\)](#)

[构造函数设计](#)

[析构函数\(C# 编程指南\)](#)

[static\(C# 参考\)](#)

使用构造函数 (C# 编程指南)

构造函数是在创建给定类型的对象时执行的类方法。构造函数具有与类相同的名称，它通常初始化新对象的数据成员。

在下面的示例中，定义了一个具有一个简单的构造函数，名为 Taxi 的类。然后使用 new 运算符来实例化该类。在为新对象分配内存之后，new 运算符立即调用 Taxi 构造函数。

C#

```
public class Taxi
{
    public bool IsInitialized;
    public Taxi()
    {
        IsInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        System.Console.WriteLine(t.IsInitialized);
    }
}
```

不带参数的构造函数称为“默认构造函数”。无论何时，只要使用 new 运算符实例化对象，并且不为 new 提供任何参数，就会调用默认构造函数。有关更多信息，请参见[实例构造函数](#)。

除非类是 static 的，否则 C# 编译器将为无构造函数的类提供一个公共的默认构造函数，以便该类可以实例化。有关更多信息，请参见[静态类和静态类成员](#)。

通过将构造函数设置为私有构造函数，可以阻止类被实例化，如下所示：

C#

```
class NLog
{
    // Private Constructor:
    private NLog() { }

    public static double E = System.Math.E; //2.71828...
}
```

有关更多信息，请参见[私有构造函数](#)。

[结构](#)类型的构造函数与类的构造函数类似，但是 [structs](#) 不能包含显式默认构造函数，因为编译器将自动提供一个构造函数。此构造函数将结构中的每个字段初始化为[默认值表](#)中显示的默认值。然而，只有当结构用 new 实例化时，才会调用此默认构造函数。例如，下面的代码使用 Int32 的默认构造函数，因此您可以确信整数已初始化：

```
int i = new int();
Console.WriteLine(i);
```

然而，下面的代码却导致了[编译器错误 CS0165](#)，因为它没有使用 new，而且试图使用尚未初始化的对象：

```
int i;
Console.WriteLine(i);
```

基于 [structs](#) 的对象可以初始化或赋值后使用，如下所示：

```
int a = 44; // Initialize the value type...
int b;
b = 33; // Or assign it before using it.
Console.WriteLine("{0}, {1}", a, b);
```

因此对值类型调用默认构造函数不是必需的。

类和 **structs** 都可以定义具有参数的构造函数。带参数的构造函数必须通过 **new** 语句或 **base** 语句来调用。类和 **structs** 还可以定义多个构造函数，并且二者均不需要定义默认构造函数。例如：

C#

```
public class Employee
{
    public int salary;

    public Employee(int annualSalary)
    {
        salary = annualSalary;
    }

    public Employee(int weeklySalary, int numberOfWeeks)
    {
        salary = weeklySalary * numberOfWeeks;
    }
}
```

此类可以使用下列语句中的任一个来创建：

C#

```
Employee e1 = new Employee(30000);
Employee e2 = new Employee(500, 52);
```

构造函数可以使用 **base** 关键字来调用基类的构造函数。例如：

C#

```
public class Manager : Employee
{
    public Manager(int annualSalary)
        : base(annualSalary)
    {
        //Add further instructions here.
    }
}
```

在此示例中，基类的构造函数在执行构造函数块之前被调用。**base** 关键字可带参数使用，也可不带参数使用。构造函数的任何参数都可用作 **base** 的参数，或用作表达式的一部分。有关更多信息，请参见 [base](#)。

在派生类中，如果不使用 **base** 关键字来显式调用基类构造函数，则将隐式调用默认构造函数（如果有的话）。这意味着下面的构造函数声明在效果上是相同的：

C#

```
public Manager(int initialdata)
{
    //Add further instructions here.
}
```

C#

```
public Manager(int initialdata) : base()
```

```
{  
    //Add further instructions here.  
}
```

如果基类没有提供默认构造函数，派生类必须使用 **base** 显式调用基构造函数。

构造函数可以使用 **this** 关键字调用同一对象中的另一构造函数。和 **base** 一样，**this** 可带参数使用也可不带参数使用，构造函数中的任何参数都可用作 **this** 的参数，或者用作表达式的一部分。例如，可以使用 **this** 重写前一示例中的第二个构造函数：

C#

```
public Employee(int weeklySalary, int numberOfWeeks)  
    : this(weeklySalary * numberOfWeeks)  
{  
}
```

上面对 **this** 关键字的使用导致此构造函数被调用：

C#

```
public Employee(int annualSalary)  
{  
    salary = annualSalary;  
}
```

构造函数可以标记为 **public**、**private**、**protected**、**internal** 或 **protectedinternal**。这些访问修饰符定义类的用户构造该类的方式。有关更多信息，请参见[访问修饰符](#)。

使用 **static** 关键字可以将构造函数声明为静态构造函数。在访问任何静态字段之前，都将自动调用静态构造函数，它们通常用于初始化静态类成员。有关更多信息，请参见[静态构造函数](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.6.6.1 构造函数
- 10.10 实例构造函数(类)
- 10.11 实例构造函数(类)
- 11.3.8 构造函数(结构)
- 11.3.10 静态构造函数(结构)

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[析构函数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[构造函数\(C# 编程指南\)](#)

实例构造函数(C# 编程指南)

实例构造函数用于创建和初始化实例。创建新对象时将调用类构造函数，例如：

C#

```
class CoOrds
{
    public int x, y;

    // constructor
    public CoOrds()
    {
        x = 0;
        y = 0;
    }
}
```

注意

为了清楚起见，此类包含公共数据成员。建议不要使用这种编程方法，因为它使程序中任何位置的任何方法都可以不受限制、不经验证地访问对象的内部组件。数据成员通常应当为私有的，并且只应当通过类方法和属性来访问。

无论何时创建基于 `CoOrds` 类的对象，都会调用此构造函数。诸如此类不带参数的构造函数称为“默认构造函数”。然而，提供其他构造函数通常十分有用。例如，可以向 `CoOrds` 类添加构造函数，以便可以为数据成员指定初始值：

C#

```
// A constructor with two arguments:
public CoOrds(int x, int y)
{
    this.x = x;
    this.y = y;
}
```

这样便可以用默认或特定的初始值创建 `CoOrds` 对象，如下所示：

C#

```
CoOrds p1 = new CoOrds();
CoOrds p2 = new CoOrds(5, 3);
```

如果类没有默认构造函数，将自动生成一个构造函数，并且将用默认值来初始化对象字段，例如，`int` 被初始化为 0。有关默认值的更多信息，请参见[默认值表\(C# 参考\)](#)。因此，由于 `CoOrds` 类的默认构造函数将所有数据成员都初始化为零，因此可以将它完全移除，而不会更改类的工作方式。本主题的稍后部分的示例 1 中提供了使用多个构造函数的完整示例，示例 2 中提供了自动生成的构造函数的示例。

也可以用实例构造函数来调用基类的实例构造函数。类构造函数可通过初始值设定项来调用基类的构造函数，如下所示：

C#

```
class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }
}
```

在此示例中，`Circle` 类将表示半径和高度的值传递给 `Shape`(`Circle` 从它派生而来)提供的构造函数。使用 `Shape` 和 `Circle` 的

完整示例请见本主题中的示例 3。

示例 1

下面的示例说明包含两个类构造函数的类：一个类构造函数没有参数，另一个类构造函数带有两个参数。

C#

```
class CoOrds
{
    public int x, y;

    // Default constructor:
    public CoOrds()
    {
        x = 0;
        y = 0;
    }

    // A constructor with two arguments:
    public CoOrds(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    // Override the ToString method:
    public override string ToString()
    {
        return (System.String.Format("{0},{1}", x, y));
    }
}

class MainClass
{
    static void Main()
    {
        CoOrds p1 = new CoOrds();
        CoOrds p2 = new CoOrds(5, 3);

        // Display the results using the overridden ToString method:
        System.Console.WriteLine("CoOrds #1 at {0}", p1);
        System.Console.WriteLine("CoOrds #2 at {0}", p2);
    }
}
```

输出

```
CoOrds #1 at (0,0)  
CoOrds #2 at (5,3)
```

示例 2

在此示例中，类 Person 没有任何构造函数；在这种情况下，将自动提供默认构造函数，同时将字段初始化为它们的默认值。

C#

```
public class Person
{
    public int age;
    public string name;
}

class TestPerson
{
    static void Main()
    {
        Person p = new Person();
```

```
        System.Console.WriteLine("Name: {0}, Age: {1}", p.name, p.age);
    }
}
```

输出

```
Name: , Age: 0
```

注意, `age` 的默认值为 0, `name` 的默认值为 `null`。有关默认值的更多信息, 请参见[默认值表\(C# 参考\)](#)。

示例 3

下面的示例说明使用基类初始值设定项。`Circle` 类是从通用类 `Shape` 派生的, `Cylinder` 类是从 `Circle` 类派生的。每个派生类的构造函数都使用其基类的初始值设定项。

C#

```
abstract class Shape
{
    public const double pi = System.Math.PI;
    protected double x, y;

    public Shape(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public abstract double Area();
}

class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }

    public override double Area()
    {
        return pi * x * x;
    }
}

class Cylinder : Circle
{
    public Cylinder(double radius, double height)
        : base(radius)
    {
        y = height;
    }

    public override double Area()
    {
        return (2 * base.Area()) + (2 * pi * x * y);
    }
}

class TestShapes
{
    static void Main()
    {
        double radius = 2.5;
        double height = 3.0;

        Circle ring = new Circle(radius);
        Cylinder tube = new Cylinder(radius, height);

        System.Console.WriteLine("Area of the circle = {0:F2}", ring.Area());
    }
}
```

```
        System.Console.WriteLine("Area of the cylinder = {0:F2}", tube.Area());  
    }  
}
```

输出

Area of the circle = 19.63

Area of the cylinder = 86.39

有关调用基类构造函数的更多示例，请参见 [virtual\(C# 参考\)](#)、[override\(C# 参考\)](#) 和 [base\(C# 参考\)](#)。

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[析构函数\(C# 编程指南\)](#)

[static\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[构造函数\(C# 编程指南\)](#)

私有构造函数(C# 编程指南)

私有构造函数是一种特殊的实例构造函数。它通常用在只包含静态成员的类中。如果类具有一个或多个私有构造函数而没有公共构造函数，则不允许其他类(除了嵌套类)创建该类的实例。例如：

C#

```
class NLog
{
    // Private Constructor:
    private NLog() { }

    public static double e = System.Math.E; //2.71828...
}
```

声明空构造函数可阻止自动生成默认构造函数。注意，如果您不对构造函数使用访问修饰符，则在默认情况下它仍为私有构造函数。但是，通常显式地使用 `private` 修饰符来清楚地表明该类不能被实例化。

当没有实例字段或实例方法(如 `Math` 类)时或者当调用方法以获得类的实例时，私有构造函数可用于阻止创建类的实例。如果类中的所有方法都是静态的，可考虑使整个类成为静态的。有关更多信息，请参见[静态类和静态类成员](#)。

示例

下面是使用私有构造函数的类的示例。

C#

```
public class Counter
{
    private Counter() { }
    public static int currentCount;
    public static int IncrementCount()
    {
        return ++currentCount;
    }
}

class TestCounter
{
    static void Main()
    {
        // If you uncomment the following statement, it will generate
        // an error because the constructor is inaccessible:
        // Counter aCounter = new Counter(); // Error

        Counter.currentCount = 100;
        Counter.IncrementCount();
        System.Console.WriteLine("New count: {0}", Counter.currentCount);
    }
}
```

输出

New count: 101

注意，如果您取消注释该示例中的以下语句，它将生成一个错误，因为该构造函数受其保护级别的限制而不可访问：

C#

```
// Counter aCounter = new Counter(); // Error
```

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 10.10.5 私有构造函数

- 25.2 静态类

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[析构函数\(C# 编程指南\)](#)

[private\(C# 参考\)](#)

[public\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[构造函数\(C# 编程指南\)](#)

静态构造函数(C# 编程指南)

静态构造函数用于初始化任何[静态](#)数据，或用于执行仅需执行一次的特定操作。在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数。

C#

```
class SimpleClass
{
    // Static constructor
    static SimpleClass()
    {
        //...
    }
}
```

静态构造函数具有以下特点：

- 静态构造函数既没有访问修饰符，也没有参数。
- 在创建第一个实例或引用任何静态成员之前，将自动调用静态构造函数来初始化类。
- 无法直接调用静态构造函数。
- 在程序中，用户无法控制何时执行静态构造函数。
- 静态构造函数的典型用途是：当类使用日志文件时，将使用这种构造函数向日志文件中写入项。
- 静态构造函数在为非托管代码创建包装类时也很有用，此时该构造函数可以调用 **LoadLibrary** 方法。

示例

在此示例中，类 `Bus` 有一个静态构造函数和一个静态成员 `Drive()`。当调用 `Drive()` 时，将调用静态构造函数来初始化类。

C#

```
public class Bus
{
    // Static constructor:
    static Bus()
    {
        System.Console.WriteLine("The static constructor invoked.");
    }

    public static void Drive()
    {
        System.Console.WriteLine("The Drive method invoked.");
    }
}

class TestBus
{
    static void Main()
    {
        Bus.Drive();
    }
}
```

输出

The static constructor invoked.

The Drive method invoked.

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[析构函数\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[构造函数\(C# 编程指南\)](#)

如何：编写复制构造函数 (C# 编程指南)

与有些语言不同，C# 不提供复制构造函数。如果您创建了新的对象并希望从现有对象复制值，您必须自行编写适当的方法。

示例

在本示例中，**Person**类包含一个构造函数，该构造函数接受另一个 **Person** 类型的对象作为参数。然后此对象的字段中的内容将分配给新对象中的字段。

C#

```

class Person
{
    private string name;
    private int age;

    // Copy constructor.
    public Person(Person previousPerson)
    {
        name = previousPerson.name;
        age = previousPerson.age;
    }

    // Instance constructor.
    public Person(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Get accessor.
    public string Details
    {
        get
        {
            return name + " is " + age.ToString();
        }
    }
}

class TestPerson
{
    static void Main()
    {
        // Create a new person object.
        Person person1 = new Person("George", 40);

        // Create another new object, copying person1.
        Person person2 = new Person(person1);
        System.Console.WriteLine(person2.Details);
    }
}

```

输出

George is 40

请参见

参考

[对象、类和结构 \(C# 编程指南\)](#)

[析构函数 \(C# 编程指南\)](#)

[ICloneable](#)

概念

[C# 编程指南](#)

[其他资源](#)

析构函数 (C# 编程指南)

析构函数用于析构类的实例。

备注

- 不能在结构中定义析构函数。只能对类使用析构函数。
- 一个类只能有一个析构函数。
- 无法继承或重载析构函数。
- 无法调用析构函数。它们是被自动调用的。
- 析构函数既没有修饰符，也没有参数。

例如，下面是类 Car 的析构函数的声明：

C#

```
class Car
{
    ~Car() // destructor
    {
        // cleanup statements...
    }
}
```

该析构函数隐式地对对象的基类调用 [Finalize](#)。这样，前面的析构函数代码被隐式地转换为：

```
protected override void Finalize()
{
    try
    {
        // cleanup statements...
    }
    finally
    {
        base.Finalize();
    }
}
```

这意味着对继承链中的所有实例递归地调用 [Finalize](#) 方法(从派生程度最大的到派生程度最小的)。

注意

不应使用空析构函数。如果类包含析构函数，[Finalize](#) 队列中则会创建一个项。调用析构函数时，将调用垃圾回收器来处理该队列。如果析构函数为空，则只会导致不必要的性能丢失。

程序员无法控制何时调用析构函数，因为这是由垃圾回收器决定的。垃圾回收器检查是否存在应用程序不再使用的对象。如果垃圾回收器认为某个对象符合析构，则调用析构函数(如果有)并回收用来存储此对象的内存。程序退出时也会调用析构函数。

可以通过调用 [Collect](#) 强制进行垃圾回收，但大多数情况下应避免这样做，因为这样会导致性能问题。有关更多信息，请参见[强制垃圾回收](#)。

使用析构函数释放资源

通常，与运行时不进行垃圾回收的编程语言相比，C# 无需太多的内存管理。这是因为 .NET Framework 垃圾回收器会隐式地管理对象的内存分配和释放。但是，当应用程序封装窗口、文件和网络连接这类非托管资源时，应当使用析构函数释放这些资源。当对象符合析构时，垃圾回收器将运行对象的 [Finalize](#) 方法。

资源的显式释放

如果您的应用程序在使用昂贵的外部资源，则还建议您提供一种在垃圾回收器释放对象前显式地释放资源的方式。可通过实现来自 [IDisposable](#) 接口的 [Dispose](#) 方法来完成这一点，该方法为对象执行必要的清理。这样可大大提高应用程序的性能。即使有

这种对资源的显式控制，析构函数也是一种保护措施，可用来在对 `Dispose` 方法的调用失败时清理资源。

有关清理资源的更多详细信息，请参见下列主题：

- [清理非托管资源](#)
- [实现 `Dispose` 方法](#)
- [using 语句\(C# 参考\)](#)

示例

下面的示例创建三个类，这三个类构成了一个继承链。类 `First` 是基类，`Second` 是从 `First` 派生的，而 `Third` 是从 `Second` 派生的。这三个类都有析构函数。在 `Main()` 中，创建了派生程度最大的类的实例。注意：程序运行时，这三个类的析构函数将自动被调用，并且是按照从派生程度最大的到派生程度最小的次序调用。

C#

```
class First
{
    ~First()
    {
        System.Console.WriteLine("First's destructor is called");
    }
}

class Second: First
{
    ~Second()
    {
        System.Console.WriteLine("Second's destructor is called");
    }
}

class Third: Second
{
    ~Third()
    {
        System.Console.WriteLine("Third's destructor is called");
    }
}

class TestDestructors
{
    static void Main()
    {
        Third t = new Third();
    }
}
```

输出

```
Third's destructor is called
Second's destructor is called
First's destructor is called
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.6 析构函数
- 10.2.7.4 为析构函数保留的成员名称
- 10.12 析构函数(类)
- 11.3.9 析构函数(结构)

请参见

[参考](#)

[IDisposable](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[构造函数 \(C# 编程指南\)](#)

[垃圾回收](#)

字段 (C# 编程指南)

本主题描述“字段”，它是包含在[类或结构](#)中的对象或值。字段使类和结构可以封装数据。

为了简单起见，这些示例使用 **public** 字段，但是不建议这样做。字段通常应为 **private**。外部类应当通过方法、属性或索引器来间接访问字段。有关更多信息，请参见[方法、属性和索引器](#)。

字段

字段存储类要满足其设计所需要的数据。例如，表示日历日期的类可能有三个整数字段：一个表示月份，一个表示日期，还有一个表示年份。在类块中声明字段的方式如下：指定字段的访问级别，然后指定字段的类型，最后指定字段的名称。例如：

C#

```
public class CalendarDate
{
    public int month;
    public int day;
    public int year;
}
```

访问对象中的字段是通过在对象名称后面依次添加一个句点和该字段的名称来实现的，具体形式为 `objectname.fieldname`。例如：

C#

```
CalendarDate birthday = new CalendarDate();
birthday.month = 7;
```

声明字段时可以使用赋值运算符为字段指定一个初始值。例如，若要自动将 7 赋给 `month` 字段，需要按如下方式声明 `month`：

C#

```
public class CalendarDateWithInitialization
{
    public int month = 7;
    //...
}
```

字段恰好在调用对象实例的构造函数之前初始化，所以，如果构造函数为字段分配了值，则它将改写字段声明期间给定的任何值。有关更多信息，请参见[使用构造函数](#)。

注意

字段初始值设定项不能引用其他实例字段。

字段可标记为 **public**、**private**、**protected**、**internal** 或 **protected internal**。这些访问修饰符定义类的使用者访问字段的方式。有关更多信息，请参见[访问修饰符](#)。

可以选择将字段声明为 **static**。这使得调用方在任何时候都能使用字段，即使类没有任何实例。有关更多信息，请参见[静态类和静态类成员](#)。

可以将字段声明为 **readonly**。只读字段只能在初始化期间或在构造函数中赋值。**static readonly** 字段非常类似于常数，只不过 C# 编译器不能在编译时访问静态只读字段的值，而只能在运行时访问。有关更多信息，请参见[常量](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.6.4 字段
- 10.4 字段

[请参见](#)

[参考](#)

[对象、类和结构\(C# 编程指南\)](#)

[使用构造函数\(C# 编程指南\)](#)

[继承\(C# 编程指南\)](#)

[访问修饰符\(C# 编程指南\)](#)

[抽象类、密封类及类成员\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

常量(C# 编程指南)

类和结构可以将常数声明为成员。常数是在编译时已知并保持不变的值。(若要创建在运行时初始化的常数值, 请使用 `readonly` 关键字。) 常数被声明为字段, 声明时在字段的类型前面使用 `const` 关键字。常数必须在声明时初始化。例如:

C#

```
class Calendar1
{
    public const int months = 12;
}
```

在此示例中, 常数 `months` 将始终为 12, 不能更改 — 即使是该类自身也不能更改它。常数必须属于整型 (`sbyte`、`byte`、`short`、`ushort`、`int`、`uint`、`long`、`ulong`、`char`、`float`、`double`、`decimal`、`bool` 或 `string`)、枚举或对 `null` 的引用。

可以同时声明多个相同类型的常数, 例如:

C#

```
class Calendar2
{
    const int months = 12, weeks = 52, days = 365;
}
```

只要不会造成循环引用, 用于初始化一个常数的表达式就可以引用另一个常数。例如:

C#

```
class Calendar3
{
    const int months = 12;
    const int weeks = 52;
    const int days = 365;

    const double daysPerWeek = days / weeks;
    const double daysPerMonth = days / months;
}
```

常数可标记为 `public`、`private`、`protected`、`internal` 或 `protectedinternal`。这些访问修饰符定义类的用户访问该常数的方式。有关更多信息, 请参见 [访问修饰符\(C# 编程指南\)](#)。

尽管常数不能使用 `static` 关键字, 但可以像访问 `静态` 字段一样访问常数。未包含在定义常数的类中的表达式必须使用类名、一个句点和常数名来访问该常数。例如:

C#

```
int birthstones = Calendar.months;
```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 10.3 常数

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

[数据类型\(C# 编程指南\)](#)

[readonly\(C# 参考\)](#)

概念

C# 编程指南

嵌套类型(C# 编程指南)

在类或结构内部定义的类型称为嵌套类型。例如：

C#

```
class Container
{
    class Nested
    {
        Nested() { }
    }
}
```

不管外部类型是类还是结构，嵌套类型均默认为 `private`，但是可以设置为 `public`、`protected internal`、`protected`、`internal` 或 `private`。在上面的示例中，`Nested` 对外部类型是不可访问的，但可以设置为 `public`，如下所示：

C#

```
class Container
{
    public class Nested
    {
        Nested() { }
    }
}
```

嵌套类型(或内部类型)可访问包含类型(或外部类型)。若要访问包含类型，请将其作为构造函数传递给嵌套类型。例如：

C#

```
public class Container
{
    public class Nested
    {
        private Container m_parent;

        public Nested()
        {
        }

        public Nested(Container parent)
        {
            m_parent = parent;
        }
    }
}
```

嵌套类型可访问包含类型的私有成员和受保护的成员(包括所有继承的私有成员或受保护的成员)。

在前面的声明中，类 `Nested` 的完整名称为 `Container.Nested`。这是用来创建嵌套类的新实例的名称，如下所示：

C#

```
Container.Nested nest = new Container.Nested();
```

请参见

参考

[对象、类和结构\(C# 编程指南\)](#)

[访问修饰符\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[构造函数 \(C# 编程指南\)](#)

访问修饰符 (C# 编程指南)

可以限制类和结构，以便只有声明它们的程序或命名空间才能使用它们。可以限制类成员，以便只有派生类才能使用它们，或者限制类成员，以便只有当前命名空间或程序中的类才能使用它们。访问修饰符是添加到类、结构或成员声明的关键字，用以指定这些限制。这些关键字包括 `public`、`private`、`protected` 和 `internal`。例如：

C#

```
public class Bicycle
{
    public void Pedal() { }
```

类和结构的可访问性

没有嵌套在其他类或结构中的类和结构可以是公共的，也可以是内部的。声明为公共的类型可由任何其他类型访问。声明为内部的类型只能由同一程序集中的类型访问。类和结构默认声明为内部的，除非向类定义添加了关键字 `public`，如前面的示例所示。类或结构定义可以添加 `internal` 关键字，使其访问级别成为显式的。访问修饰符不影响类或结构自身 -- 它始终能够访问自身及其所有成员。

类成员和结构成员的可访问性

可以使用五种访问类型之一来声明类成员或结构成员。与类和结构自身一样，它们也可以是公共的或内部的。可以使用 `protected` 关键字将类成员声明为受保护的，意味着只有使用该类作为基类的派生类型才能访问该成员。通过组合 `protected` 和 `internal` 关键字，可以将类成员标记为受保护的内部成员 -- 只有派生类型或同一程序集中的类型才能访问该成员。最后，可以使用 `private` 关键字将类成员或结构成员声明为私有的，指示只有声明该成员的类或结构才能访问该成员。

若要设置类成员或结构成员的访问级别，请向该成员声明添加适当的关键字。示例：

C#

```
// public class:
public class Tricycle
{
    // protected method:
    protected void Pedal() { }

    // private field:
    private int m_wheels = 3;

    // protected internal property:
    protected internal int Wheels
    {
        get { return m_wheels; }
    }
}
```

其他类型

与类一样，接口也可声明为公共类型或内部类型。与类不同，接口默认具有内部访问级别。接口成员始终是公共的，不能应用任何访问修饰符。

命名空间和枚举成员始终是公共的，不能应用任何访问修饰符。

委托默认具有内部访问级别。

默认情况下，在命名空间中或在编译单元顶部（例如，不在命名空间、类或结构中）声明的任何类型都是内部的，但是可以成为公共的。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.2.3 访问修饰符

[请参见](#)

[参考](#)

[对象、类和结构\(C# 编程指南\)](#)

[接口\(C# 编程指南\)](#)

[private\(C# 参考\)](#)

[public\(C# 参考\)](#)

[internal\(C# 参考\)](#)

[protected\(C# 参考\)](#)

[class\(C# 参考\)](#)

[struct\(C# 参考\)](#)

[接口\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

分部类定义(C# 编程指南)

可以将类、结构或接口的定义拆分到两个或多个源文件中。每个源文件包含类定义的一部分，编译应用程序时将把所有部分组合起来。在以下几种情况下需要拆分类定义：

- 处理大型项目时，使一个类分布于多个独立文件中可以让多位程序员同时对该类进行处理。
- 使用自动生成的源时，无需重新创建源文件便可将代码添加到类中。Visual Studio 在创建 Windows 窗体、Web 服务包装代码等时都使用此方法。您无需编辑 Visual Studio 所创建的文件，便可创建使用这些类的代码。
- 若要拆分类定义，请使用 `partial` 关键字修饰符，如下所示：

C#

```
public partial class Employee
{
    public void DoWork()
    {
    }
}

public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

备注

使用 `partial` 关键字表明可在命名空间内定义该类、结构或接口的其他部分。所有部分都必须使用 `partial` 关键字。在编译时，各个部分都必须可用来形成最终的类型。各个部分必须具有相同的可访问性，如 `public`、`private` 等。

如果将任意部分声明为抽象的，则整个类型都被视为抽象的。如果将任意部分声明为密封的，则整个类型都被视为密封的。如果将任意部分声明为基类型，则整个类型都将继承该类。

指定基类的所有部分必须一致，但忽略基类的部分仍继承该基类型。各个部分可以指定不同的基接口，最终类型将实现所有部分声明所列出的全部接口。在某一分部定义中声明的任何类、结构或接口成员可供所有其他部分使用。最终类型是所有部分在编译时的组合。

注意

分部修饰符不可用于委托或枚举声明中。

嵌套类型可以是分部的，即使它们所嵌套于的类型本身并不是分部的也如此。例如：

C#

```
class Container
{
    partial class Nested
    {
        void Test() { }
    }
    partial class Nested
    {
        void Test2() { }
    }
}
```

- 编译时将对分部类型定义的属性进行合并。例如，下面的声明：

C#

```
[System.SerializableAttribute]  
partial class Moon { }
```

```
[System.ObsoleteAttribute]  
partial class Moon { }
```

等效于：

C#

```
[System.SerializableAttribute]  
[System.ObsoleteAttribute]  
class Moon { }
```

- 将从所有分部类型定义中对以下内容进行合并：
 - XML 注释
 - interfaces
 - 泛型类型参数属性
 - 类属性
 - 成员

例如，下面的声明：

C#

```
partial class Earth : Planet, IRotate { }  
partial class Earth : IRevolve { }
```

等效于：

C#

```
class Earth : Planet, IRotate, IRevolve { }
```

限制

处理分部类定义时需遵循下面的几个规则：

- 要作为同一类型的各个部分的所有分部类型定义都必须使用 **partial** 进行修饰。例如，下面的类声明将生成错误：

C#

```
public partial class A { }  
//public class A { } // Error, must also be marked partial
```

- **partial** 修饰符只能出现在紧靠关键字 **class**、**struct** 或 **interface** 前面的位置。

- 分部类型定义中允许使用嵌套的分部类型，例如：

C#

```
partial class ClassWithNestedClass  
{  
    partial class NestedClass { }  
}  
  
partial class ClassWithNestedClass
```

```
{  
    partial class NestedClass { }  
}
```

- 要成为同一类型的各个部分的所有分部类型定义都必须在同一程序集和同一模块(.exe 或 .dll 文件)中进行定义。分部定义不能跨越多个模块。
- 类名和泛型类型参数在所有的分部类型定义中都必须匹配。泛型类型可以是分部的。每个分部声明都必须以相同的顺序使用相同的参数名。
- 下面的用于分部类型定义中的关键字是可选的，但是如果某关键字出现在一个分部类型定义中，则该关键字不能与在同一类型的其他分部定义中指定的关键字冲突：

- public
- private
- protected
- internal
- abstract
- sealed
- 基类
- new 修饰符(嵌套部分)
- 一般约束(有关更多信息，请参见[类型参数的约束\(C# 编程指南\)](#)。)

示例 1

下面的示例在一个分部类定义中声明类 CoOrds 的字段和构造函数，在另一个分部类定义中声明成员 PrintCoOrds。

C#

```
public partial class CoOrds  
{  
    private int x;  
    private int y;  
  
    public CoOrds(int x, int y)  
    {  
        this.x = x;  
        this.y = y;  
    }  
}  
  
public partial class CoOrds  
{  
    public void PrintCoOrds()  
    {  
        System.Console.WriteLine("CoOrds: {0},{1}", x, y);  
    }  
}  
  
class TestCoOrds  
{  
    static void Main()  
    {  
        CoOrds myCoOrds = new CoOrds(10, 15);  
        myCoOrds.PrintCoOrds();  
    }  
}
```

输出

CoOrds: 10,15

示例 2

从下面的示例可以看出，您也可以开发分部结构和接口。

C#

```
partial interface ITest
{
    void Interface_Test();
}

partial interface ITest
{
    void Interface_Test2();
}

partial struct S1
{
    void Struct_Test() { }
}

partial struct S1
{
    void Struct_Test2() { }
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 23 个分部类型

请参见

参考

[接口 \(C# 编程指南\)](#)

[partial \(C# 参考\)](#)

概念

[C# 编程指南](#)

[类 \(C# 编程指南\)](#)

[结构 \(C# 编程指南\)](#)

静态类和静态类成员 (C# 编程指南)

静态类和类成员用于创建无需创建类的实例就能够访问的数据和函数。静态类成员可用于分离独立于任何对象标识的数据和行为：无论对象发生什么更改，这些数据和函数都不会随之变化。当类中没有依赖对象标识的数据或行为时，就可以使用静态类。

静态类

类可以声明为 `static` 的，以指示它仅包含静态成员。不能使用 `new` 关键字创建静态类的实例。静态类在加载包含该类的程序或命名空间时由 .NET Framework 公共语言运行库 (CLR) 自动加载。

使用静态类来包含不与特定对象关联的方法。例如，创建一组不操作实例数据并且不与代码中的特定对象关联的方法是很常见的要求。您应该使用静态类来包含那些方法。

静态类的主要功能如下：

- 它们仅包含静态成员。
- 它们不能被实例化。
- 它们是密封的。
- 它们不能包含[实例构造函数 \(C# 编程指南\)](#)。

因此创建静态类与创建仅包含静态成员和私有构造函数的类大致一样。私有构造函数阻止类被实例化。

使用静态类的优点在于，编译器能够执行检查以确保不致偶然地添加实例成员。编译器将保证不会创建此类的实例。

静态类是密封的，因此不可被继承。静态类不能包含构造函数，但仍可声明静态构造函数以分配初始值或设置某个静态状态。有关更多信息，请参见[静态构造函数 \(C# 编程指南\)](#)。

何时使用静态类

假设有一个类 `CompanyInfo`，它包含用于获取有关公司名称和地址信息的下列方法。

C#

```
class CompanyInfo
{
    public string GetCompanyName() { return "CompanyName"; }
    public string GetCompanyAddress() { return "CompanyAddress"; }
    //...
}
```

不需要将这些方法附加到该类的具体实例。因此，您可以将它声明为静态类，而不是创建此类的不必要的实例，如下所示：

C#

```
static class CompanyInfo
{
    public static string GetCompanyName() { return "CompanyName"; }
    public static string GetCompanyAddress() { return "CompanyAddress"; }
    //...
}
```

使用静态类作为不与特定对象关联的方法的组织单元。此外，静态类能够使您的实现更简单、迅速，因为您不必创建对象就能调用其方法。以一种有意义的方式组织类内部的方法（例如 `System` 命名空间中的 `Math` 类的方法）是很有用的。

静态成员

即使没有创建类的实例，也可以调用该类中的静态方法、字段、属性或事件。如果创建了该类的任何实例，不能使用实例来访问静态成员。只存在静态字段和事件的一个副本，静态方法和属性只能访问静态字段和静态事件。静态成员通常用于表示不会随对象状态而变化的数据或计算；例如，数学库可能包含用于计算正弦和余弦的静态方法。

在成员的返回类型之前使用 `static` 关键字来声明静态类成员，例如：

C#

```

public class Automobile
{
    public static int NumberOfWheels = 4;
    public static int SizeOfGasTank
    {
        get
        {
            return 15;
        }
    }
    public static void Drive() { }
    public static event EventType RunOutOfGas;

    //other non-static fields and properties...
}

```

静态成员在第一次被访问之前并且在任何静态构造函数(如调用的话)之前初始化。若要访问静态类成员，应使用类名而不是变量名来指定该成员的位置。例如：

C#

```

Automobile.Drive();
int i = Automobile.NumberOfWheels;

```

示例

下面是一个静态类的示例，它包含两个在摄氏温度和华氏温度之间执行来回转换的方法：

C#

```

public static class TemperatureConverter
{
    public static double CelsiusToFahrenheit(string temperatureCelsius)
    {
        // Convert argument to double for calculations.
        double celsius = System.Double.Parse(temperatureCelsius);

        // Convert Celsius to Fahrenheit.
        double fahrenheit = (celsius * 9 / 5) + 32;

        return fahrenheit;
    }

    public static double FahrenheitToCelsius(string temperatureFahrenheit)
    {
        // Convert argument to double for calculations.
        double fahrenheit = System.Double.Parse(temperatureFahrenheit);

        // Convert Fahrenheit to Celsius.
        double celsius = (fahrenheit - 32) * 5 / 9;

        return celsius;
    }
}

class TestTemperatureConverter
{
    static void Main()
    {
        System.Console.WriteLine("Please select the convertor direction");
        System.Console.WriteLine("1. From Celsius to Fahrenheit.");
        System.Console.WriteLine("2. From Fahrenheit to Celsius.");
        System.Console.Write(":");

        string selection = System.Console.ReadLine();
    }
}

```

```
double F, C = 0;

switch (selection)
{
    case "1":
        System.Console.Write("Please enter the Celsius temperature: ");
        F = TemperatureConverter.CelsiusToFahrenheit(System.Console.ReadLine());
        System.Console.WriteLine("Temperature in Fahrenheit: {0:F2}", F);
        break;

    case "2":
        System.Console.Write("Please enter the Fahrenheit temperature: ");
        C = TemperatureConverter.FahrenheitToCelsius(System.Console.ReadLine());
        System.Console.WriteLine("Temperature in Celsius: {0:F2}", C);
        break;

    default:
        System.Console.WriteLine("Please select a convertor.");
        break;
}
}
```

输入

```
2
98.6
```

示例输出：

```
Please select the convertor
1. From Celsius to Fahrenheit.
2. From Fahrenheit to Celsius.

:2

Please enter the Fahrenheit temperature: 98.6
Temperature in Celsius: 37.00
```

附加的示例输出可能如下所示：

```
Please select the convertor
1. From Celsius to Fahrenheit.
2. From Fahrenheit to Celsius.

:1

Please enter the Celsius temperature: 37.00
Temperature in Fahrenheit: 98.60
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 25.2 静态类

请参见

参考

[class\(C# 参考\)](#)

[实例构造函数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[静态类设计](#)

[类\(C# 编程指南\)](#)

如何：了解向方法传递结构和向方法传递类引用之间的区别(C# 编程指南)

本示例演示在向方法传递[结构](#)时，传递的是该结构的副本，而在传递[类实例](#)时，传递的是一个引用。

本示例的输出表明：当向 ClassTaker 方法传递类实例时，只更改类字段的值。但是向 StructTaker 方法传递结构实例并不更改结构字段。这是因为向 StructTaker 方法传递的是结构的副本，而向 ClassTaker 方法传递的是对类的引用。

示例

C#

```
class TheClass
{
    public string willIChange;
}

struct TheStruct
{
    public string willIChange;
}

class TestClassAndStruct
{
    static void ClassTaker(TheClass c)
    {
        c.willIChange = "Changed";
    }

    static void StructTaker(TheStruct s)
    {
        s.willIChange = "Changed";
    }

    static void Main()
    {
        TheClass testClass = new TheClass();
        TheStruct testStruct = new TheStruct();

        testClass.willIChange = "Not Changed";
        testStruct.willIChange = "Not Changed";

        ClassTaker(testClass);
        StructTaker(testStruct);

        System.Console.WriteLine("Class field = {0}", testClass.willIChange);
        System.Console.WriteLine("Struct field = {0}", testStruct.willIChange);
    }
}
```

输出

```
Class field = Changed
Struct field = Not Changed
```

请参见

参考

[传递参数\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[类\(C# 编程指南\)](#)

[结构\(C# 编程指南\)](#)

属性 (C# 编程指南)

属性是这样的成员：它们提供灵活的机制来读取、编写或计算私有字段的值。可以像使用公共数据成员一样使用属性，但实际上它们是称为“访问器”的特殊方法。这使得数据在可被轻松访问的同时，仍能提供方法的安全性和灵活性。

在本示例中，类 `TimePeriod` 存储了一个时间段。类内部以秒为单位存储时间，但提供一个称为 `Hours` 的属性，它允许客户端指定以小时为单位的时间。`Hours` 属性的访问器执行小时和秒之间的转换。

示例

C#

```
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds / 3600; }
        set { seconds = value * 3600; }
    }
}

class Program
{
    static void Main()
    {
        TimePeriod t = new TimePeriod();

        // Assigning the Hours property causes the 'set' accessor to be called.
        t.Hours = 24;

        // Evaluating the Hours property causes the 'get' accessor to be called.
        System.Console.WriteLine("Time in hours: " + t.Hours);
    }
}
```

输出

Time in hours: 24

属性概述

- 属性使类能够以一种公开的方法获取和设置值，同时隐藏实现或验证代码。
- `get` 属性访问器用于返回属性值，而 `set` 访问器用于分配新值。这些访问器可以有不同的访问级别。有关更多信息，请参见[访问器可访问性](#)。
- `value` 关键字用于定义由 `set` 索引器分配的值。
- 不实现 `set` 方法的属性是只读的。

相关章节

- [使用属性 \(C# 编程指南\)](#)
- [接口属性 \(C# 编程指南\)](#)
- [属性和索引器之间的比较 \(C# 编程指南\)](#)
- [非对称访问器可访问性 \(C# 编程指南\)](#)
- [“属性”示例](#)

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.6.6.2 属性
- 10.2.7.1 为属性保留的成员名称
- 10.6 属性

请参见

参考

[使用属性\(C# 编程指南\)](#)

[索引器\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[属性设计](#)

使用属性 (C# 编程指南)

属性结合了字段和方法的多个方面。对于对象的用户，属性显示为字段，访问该属性需要完全相同的语法。对于类的实现者，属性是一个或两个代码块，表示一个 **get** 访问器和/或一个 **set** 访问器。当读取属性时，执行 **get** 访问器的代码块；当向属性分配一个新值时，执行 **set** 访问器的代码块。不具有 **set** 访问器的属性被视为只读属性。不具有 **get** 访问器的属性被视为只写属性。同时具有这两个访问器的属性是读写属性。

与字段不同，属性不作为变量来分类。因此，不能将属性作为 [ref\(C# 参考\)](#) 参数或 [out\(C# 参考\)](#) 参数传递。

属性具有多种用法：它们可在允许更改前验证数据；它们可透明地公开某个类上的数据，该类的数据实际上是从其他源（例如数据库）检索到的；当数据被更改时，它们可采取行动，例如引发事件或更改其他字段的值。

属性在类模块内是通过以下方式声明的：指定字段的访问级别，后面是属性的类型，接下来是属性的名称，然后是声明 **get** 访问器和/或 **set** 访问器的代码模块。例如：

C#

```
public class Date
{
    private int month = 7; // "backing store"

    public int Month
    {
        get
        {
            return month;
        }
        set
        {
            if ((value > 0) && (value < 13))
            {
                month = value;
            }
        }
    }
}
```

在本例中，`Month` 是作为属性声明的，这样 **set** 访问器可确保 `Month` 值设置为 1 和 12 之间。`Month` 属性使用私有字段来跟踪实际值。属性的数据的真实位置经常称为属性的“后备存储”。属性使用作为后备存储的私有字段是很常见的。将字段标记为私有可确保该字段只能通过调用属性来更改。有关公共和私有访问限制的更多信息，请参见[访问修饰符 \(C# 编程指南\)](#)。

get 访问器

get 访问器体与方法体相似。它必须返回属性类型的值。执行 **get** 访问器相当于读取字段的值。例如，当正在从 **get** 访问器返回私有变量并且启用了优化时，对 **get** 访问器方法的调用由编译器进行内联，因此不存在方法调用的系统开销。然而，由于在编译时编译器不知道在运行时实际调用哪个方法，无法内联虚拟 **get** 访问器。以下是返回私有字段 `name` 的值的 **get** 访问器：

C#

```
class Person
{
    private string name; // the name field
    public string Name // the Name property
    {
        get
        {
            return name;
        }
    }
}
```

当引用属性时，除非该属性为赋值目标，否则将调用 **get** 访问器以读取该属性的值。例如：

C#

```
Person p1 = new Person();
//...

System.Console.WriteLine(p1.Name); // the get accessor is invoked here
```

get 访问器必须以 `return` 或 `throw` 语句终止，并且控制权不能离开访问器体。

通过使用 **get** 访问器更改对象的状态不是一种好的编程风格。例如，以下访问器在每次访问 `number` 字段时都产生更改对象状态的副作用。

C#

```
private int number;
public int Number
{
    get
    {
        return number++; // Don't do this
    }
}
```

get 访问器可用于返回字段值，或用于计算并返回字段值。例如：

C#

```
class Employee
{
    private string name;
    public string Name
    {
        get
        {
            return name != null ? name : "NA";
        }
    }
}
```

在上述代码段中，如果不对 `Name` 属性赋值，它将返回值 NA。

set 访问器

set 访问器类似于返回类型为 `void` 的方法。它使用称为 `value` 的隐式参数，此参数的类型是属性的类型。在下面的示例中，将 **set** 访问器添加到 `Name` 属性：

C#

```
class Person
{
    private string name; // the name field
    public string Name // the Name property
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }
}
```

当对属性赋值时，用提供新值的参数调用 **set** 访问器。例如：

C#

```
Person p1 = new Person();
p1.Name = "Joe"; // the set accessor is invoked here

System.Console.WriteLine(p1.Name); // the get accessor is invoked here
```

在 **set** 访问器中，对局部变量声明使用隐式参数名称 *value* 是错误的。

备注

可将属性标记为 **public**、**private**、**protected**、**internal** 或 **protected internal**。这些访问修饰符定义类的用户如何才能访问属性。同一属性的 **get** 和 **set** 访问器可能具有不同的访问修饰符。例如，**get** 可能是 **public** 以允许来自类型外的只读访问；**set** 可能是 **private** 或 **protected**。有关更多信息，请参见[访问修饰符\(C# 编程指南\)](#)。

可以使用 **static** 关键字将属性声明为静态属性。这使得调用方随时可使用该属性，即使不存在类的实例。有关更多信息，请参见[静态类和静态类成员\(C# 编程指南\)](#)。

可以使用 **virtual** 关键字将属性标记为虚属性。这样，派生类就可以使用 **override** 关键字来重写属性行为。有关这些选项的更多信息，请参见[继承\(C# 编程指南\)](#)。

重写虚属性的属性还可以是 **sealed** 的，这表示它对派生类不再是虚拟的。最后一点，属性可以声明为 **abstract**，这意味着在类中没有实现，派生类必须编写自己的实现。有关这些选项的更多信息，请参见[抽象类、密封类及类成员\(C# 编程指南\)](#)。

注意

对 **static** 属性的访问器使用 [virtual\(C# 参考\)](#)、[abstract\(C# 参考\)](#) 或 [override\(C# 参考\)](#) 修饰符是错误的。

示例 1

此例说明了实例、静态和只读属性。它从键盘接受雇员的姓名，按 1 递增 **NumberOfEmployees**，并显示雇员的姓名和编号。

C#

```
public class Employee
{
    public static int NumberOfEmployees;
    private static int counter;
    private string name;

    // A read-write instance property:
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    // A read-only static property:
    public static int Counter
    {
        get { return counter; }
    }

    // A Constructor:
    public Employee()
    {
        // Calculate the employee's number:
        counter = ++counter + NumberOfEmployees;
    }
}

class TestEmployee
{
    static void Main()
    {
        Employee.NumberOfEmployees = 100;
        Employee e1 = new Employee();
        e1.Name = "Claude Vige";
```

```

        System.Console.WriteLine("Employee number: {0}", Employee.Counter);
        System.Console.WriteLine("Employee name: {0}", e1.Name);
    }
}

```

输出 1

```

Employee number: 101
Employee name: Claude Vige

```

示例 2

此例说明如何访问基类中被派生类中具有同一名称的另一个属性隐藏的属性。

C#

```

public class Employee
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}

public class Manager : Employee
{
    private string name;

    // Notice the use of the new modifier:
    public new string Name
    {
        get { return name; }
        set { name = value + ", Manager"; }
    }
}

class TestHiding
{
    static void Main()
    {
        Manager m1 = new Manager();

        // Derived class property.
        m1.Name = "John";

        // Base class property.
        ((Employee)m1).Name = "Mary";

        System.Console.WriteLine("Name in the derived class is: {0}", m1.Name);
        System.Console.WriteLine("Name in the base class is: {0}", ((Employee)m1).Name);
    }
}

```

输出 2

```

Name in the derived class is: John, Manager
Name in the base class is: Mary

```

代码讨论

以下是上述示例中的要点：

- 派生类中的属性 `Name` 隐藏基类中的属性 `Name`。在这种情况下，派生类的属性声明中使用 `new` 修饰符：

C#

```
public new string Name
```

- 转换 (Employee) 用于访问基类中的隐藏属性：

C#

```
((Employee)m1).Name = "Mary";
```

有关隐藏成员的更多信息，请参见 [new 修饰符 \(C# 参考\)](#)。

示例 3

在此例中，Cube 和 Square 这两个类实现抽象类 Shape，并重写它的抽象 Area 属性。注意属性上 `override` 修饰符的使用。程序接受输入的边长并计算正方形和立方体的面积。它还接受输入的面积并计算正方形和立方体的相应边长。

C#

```
abstract class Shape
{
    public abstract double Area
    {
        get;
        set;
    }
}

class Square : Shape
{
    public double side;

    public Square(double s) //constructor
    {
        side = s;
    }

    public override double Area
    {
        get
        {
            return side * side;
        }
        set
        {
            side = System.Math.Sqrt(value);
        }
    }
}

class Cube : Shape
{
    public double side;

    public Cube(double s)
    {
        side = s;
    }

    public override double Area
    {
        get
        {
            return 6 * side * side;
        }
    }
}
```

```
        set
    {
        side = System.Math.Sqrt(value / 6);
    }
}

class TestShapes
{
    static void Main()
    {
        // Input the side:
        System.Console.Write("Enter the side: ");
        double side = double.Parse(System.Console.ReadLine());

        // Compute the areas:
        Square s = new Square(side);
        Cube c = new Cube(side);

        // Display the results:
        System.Console.WriteLine("Area of the square = {0:F2}", s.Area);
        System.Console.WriteLine("Area of the cube = {0:F2}", c.Area);
        System.Console.WriteLine();

        // Input the area:
        System.Console.Write("Enter the area: ");
        double area = double.Parse(System.Console.ReadLine());

        // Compute the sides:
        s.Area = area;
        c.Area = area;

        // Display the results:
        System.Console.WriteLine("Side of the square = {0:F2}", s.side);
        System.Console.WriteLine("Side of the cube = {0:F2}", c.side);
    }
}
```

输入

```
4  
24
```

输出 3

```
Enter the side: 4
Area of the square = 16.00
Area of the cube = 96.00
Enter the area: 24
Side of the square = 4.90
Side of the cube = 2.00
```

请参见

任务

[“属性”示例](#)

参考

[属性 \(C# 编程指南\)](#)

[接口属性 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

接口属性 (C# 编程指南)

可以在[接口 \(C# 参考\)](#)上声明属性。以下是接口索引器访问器的示例：

C#

```
public interface ISampleInterface
{
    // Property declaration:
    string Name
    {
        get;
        set;
    }
}
```

接口属性的访问器不具有体。因此，访问器的用途是指示属性是否为读写、只读或只写。

示例

在此例中，接口 `IEmployee` 具有读写属性 `Name` 和只读属性 `Counter`。`Employee` 类实现 `IEmployee` 接口并使用这两种属性。程序读取新雇员的姓名和雇员的当前编号，并显示雇员姓名和计算所得的雇员编号。

可以使用属性的完全限定名，它引用声明成员的接口。例如：

C#

```
string IEmployee.Name
{
    get { return "Employee Name"; }
    set { }
}
```

这被称为[显式接口实现 \(C# 编程指南\)](#)。例如，如果 `Employee` 类实现两个接口 `ICitizen` 和 `IEmployee`，并且两个接口都具有 `Name` 属性，则需要显式接口成员实现。即，如下属性声明：

C#

```
string IEmployee.Name
{
    get { return "Employee Name"; }
    set { }
}
```

在 `IEmployee` 接口上实现 `Name` 属性，而下面的声明：

C#

```
string ICitizen.Name
{
    get { return "Citizen Name"; }
    set { }
}
```

在 `ICitizen` 接口上实现 `Name` 属性。

C#

```
interface IEmployee
{
    string Name
    {
```

```

        get;
        set;
    }

    int Counter
    {
        get;
    }
}

public class Employee : IEmployee
{
    public static int numberOfEmployees;

    private string name;
    public string Name // read-write instance property
    {
        get
        {
            return name;
        }
        set
        {
            name = value;
        }
    }

    private int counter;
    public int Counter // read-only instance property
    {
        get
        {
            return counter;
        }
    }

    public Employee() // constructor
    {
        counter = ++counter + numberOfEmployees;
    }
}

class TestEmployee
{
    static void Main()
    {
        System.Console.Write("Enter number of employees: ");
        Employee.numberOfEmployees = int.Parse(System.Console.ReadLine());

        Employee e1 = new Employee();
        System.Console.Write("Enter the name of the new employee: ");
        e1.Name = System.Console.ReadLine();

        System.Console.WriteLine("The employee information:");
        System.Console.WriteLine("Employee number: {0}", e1.Counter);
        System.Console.WriteLine("Employee name: {0}", e1.Name);
    }
}

```

输入

210
Hazem Abolrous

示例输出

Enter number of employees: 210

Enter the name of the new employee: Hazem Abolrous

The employee information:

Employee number: 211

Employee name: Hazem Abolrous

请参见

参考

[属性 \(C# 编程指南\)](#)

[使用属性 \(C# 编程指南\)](#)

[属性和索引器之间的比较 \(C# 编程指南\)](#)

[索引器 \(C# 编程指南\)](#)

[接口 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

非对称访问器可访问性(C# 编程指南)

属性或索引器的 `get` 和 `set` 部分称为“访问器”。默认情况下，这些访问器具有相同的可见性或访问级别：其所属属性或索引器的可见性或访问级别。有关更多信息，请参见[可访问性级别](#)。不过，有时限制对其中某个访问器的访问会很有用。通常是在保持 `get` 访问器可公开访问的情况下，限制 `set` 访问器的可访问性。例如：

C#

```
public string Name
{
    get
    {
        return name;
    }
    protected set
    {
        name = value;
    }
}
```

在此示例中，名为 `Name` 的属性定义了一个 `get` 访问器和一个 `set` 访问器。`get` 访问器接受该属性本身的可访问性级别（在此示例中为 `public`），而对于 `set` 访问器，则通过对该访问器本身应用 `protected` 访问修饰符来进行显式限制。

对访问器的访问修饰符的限制

对属性或索引器使用访问修饰符受以下条件的制约：

- 不能对接口或显式[接口](#)成员实现使用访问器修饰符。
- 仅当属性或索引器同时具有 `set` 和 `get` 访问器时，才能使用访问器修饰符。这种情况下，只允许对其中一个访问器使用修饰符。
- 如果属性或索引器具有 `override` 修饰符，则访问器修饰符必须与重写的访问器的访问器（如果有的话）匹配。
- 访问器的可访问性级别必须比属性或索引器本身的可访问性级别具有更严格的限制。

重写访问器的访问修饰符

在重写属性或索引器时，被重写的访问器对重写代码而言，必须是可访问的。此外，属性/索引器和访问器的可访问性级别都必须与相应的被重写属性/索引器和访问器匹配。例如：

C#

```
public class Parent
{
    public virtual int TestProperty
    {
        // Notice the accessor accessibility level.
        protected set { }

        // No access modifier is used here.
        get { return 0; }
    }
}
public class Kid : Parent
{
    public override int TestProperty
    {
        // Use the same accessibility level as in the overridden accessor.
        protected set { }

        // Cannot use access modifier here.
        get { return 0; }
    }
}
```

实现接口

使用访问器实现接口时，访问器不能具有访问修饰符。但是，如果使用一个访问器(如 **get**)实现接口，则另一个访问器可以具有访问修饰符，如下面的示例所示：

C#

```
public interface ISomeInterface
{
    int TestProperty
    {
        // No access modifier allowed here
        // because this is an interface.
        get;
    }
}

public class TestClass : ISomeInterface
{
    public int TestProperty
    {
        // Cannot use access modifier here because
        // this is an interface implementation.
        get { return 10; }

        // Interface property does not have set accessor,
        // so access modifier is allowed.
        protected set { }
    }
}
```

访问器可访问性域

如果对访问器使用访问某个修饰符，则访问器的[可访问性域](#)由该修饰符确定。

如果不对访问器使用访问修饰符，则访问器的可访问性域由属性或索引器的可访问性级别确定。

示例

下面的示例包含三个类：BaseClass、DerivedClass 和 MainClass。每个类的 BaseClass、Name 和 Id 都有两个属性。该示例演示在使用限制性访问修饰符(如 **protected** 或 **private**)时，如何通过 BaseClass 的 Id 属性隐藏 DerivedClass 的 Id 属性。因此，向该属性赋值时，将调用 BaseClass 类中的属性。将访问修饰符替换为 **public** 将使该属性可访问。

该示例还演示 DerivedClass 的 Name 属性的 set 访问器上的限制性访问修饰符(如 **private** 或 **protected**)如何防止对该访问器的访问，并在向它赋值时生成错误。

C#

```
public class BaseClass
{
    private string name = "Name-BaseClass";
    private string id = "ID-BaseClass";

    public string Name
    {
        get { return name; }
        set { }
    }

    public string Id
    {
        get { return id; }
        set { }
    }
}

public class DerivedClass : BaseClass
{
```

```

private string name = "Name-DerivedClass";
private string id = "ID-DerivedClass";

new public string Name
{
    get
    {
        return name;
    }

    // Using "protected" would make the set accessor not accessible.
    set
    {
        name = value;
    }
}

// Using private on the following property hides it in the Main Class.
// Any assignment to the property will use Id in BaseClass.
new private string Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

class MainClass
{
    static void Main()
    {
        BaseClass b1 = new BaseClass();
        DerivedClass d1 = new DerivedClass();

        b1.Name = "Mary";
        d1.Name = "John";

        b1.Id = "Mary123";
        d1.Id = "John123"; // The BaseClass.Id property is called.

        System.Console.WriteLine("Base: {0}, {1}", b1.Name, b1.Id);
        System.Console.WriteLine("Derived: {0}, {1}", d1.Name, d1.Id);
    }
}

```

输出

Name and ID in the base class: Name-BaseClass, ID-BaseClass

Name and ID in the derived class: John, ID-BaseClass

注释

注意，如果将 new private string Id 替换为 new public string Id，则得到如下输出：

Name and ID in the base class: Name-BaseClass, ID-BaseClass

Name and ID in the derived class: John, John123

请参见

参考

[属性 \(C# 编程指南\)](#)

[索引器 \(C# 编程指南\)](#)

[访问修饰符 \(C# 编程指南\)](#)

概念

C# 编程指南

如何：声明和使用读/写属性 (C# 编程指南)

属性可以提供公共数据成员的便利，而又不会带来不受保护、不受控制以及未经验证访问对象数据的风险。这是通过“访问器”来实现的：访问器是为基础数据成员赋值和检索其值的特殊方法。使用 `set` 访问器可以为数据成员赋值，使用 `get` 访问器可以检索数据成员的值。

此示例演示 `Person` 类，该类具有两个属性：`Name` (`string`) 和 `Age` (`int`)。这两个属性都提供 `get` 和 `set` 访问器，因此它们被视为读/写属性。

示例

C#

```
class Person
{
    private string m_name = "N/A";
    private int m_Age = 0;

    // Declare a Name property of type string:
    public string Name
    {
        get
        {
            return m_name;
        }
        set
        {
            m_name = value;
        }
    }

    // Declare an Age property of type int:
    public int Age
    {
        get
        {
            return m_Age;
        }

        set
        {
            m_Age = value;
        }
    }

    public override string ToString()
    {
        return "Name = " + Name + ", Age = " + Age;
    }
}

class TestPerson
{
    static void Main()
    {
        // Create a new Person object:
        Person person = new Person();

        // Print out the name and the age associated with the person:
        System.Console.WriteLine("Person details - {0}", person);

        // Set some values on the person object:
        person.Name = "Joe";
        person.Age = 99;
        System.Console.WriteLine("Person details - {0}", person);

        // Increment the Age property:
    }
}
```

```
    person.Age += 1;
    System.Console.WriteLine("Person details - {0}", person);
}
```

输出

```
Person details - Name = N/A, Age = 0
Person details - Name = Joe, Age = 99
Person details - Name = Joe, Age = 100
```

可靠编程

在上面的示例中，`Name` 和 `Age` 属性是公共的，并且同时包含 **get** 和 **set** 访问器。这允许任何对象读写这些属性。不过，有时需要排除其中的一个访问器。例如，省略 **set** 访问器将使该属性成为只读的：

C#

```
public string Name
{
    get
    {
        return m_name;
    }
}
```

此外，您还可以公开一个访问器，而使另一个访问器成为私有的或受保护的。有关更多信息，请参见[非对称访问器可访问性](#)。

声明了属性后，可像使用类的字段那样使用这些属性。这使得获取和设置属性值时都可以使用非常自然的语法，如以下语句中所示：

C#

```
person.Name = "Joe";
person.Age = 99;
```

注意，属性 **set** 方法中可以使用一个特殊的 `value` 变量。该变量包含用户指定的值，例如：

C#

```
m_name = value;
```

请注意用于使 `Person` 对象上的 `Age` 属性递增的简洁语法：

C#

```
person.Age += 1;
```

如果将单独的 **set** 和 **get** 方法用于模型属性，则等效代码可能类似于：

```
person.SetAge(person.GetAge() + 1);
```

本示例中重写了 **ToString** 方法：

C#

```
public override string ToString()
{
    return "Name = " + Name + ", Age = " + Age;
}
```

注意，程序中未显式使用 **ToString**。默认情况下，它由 **WriteLine** 调用来调用。

[请参见](#)

[参考](#)

[属性\(C# 编程指南\)](#)

[对象、类和结构\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

索引器 (C# 编程指南)

索引器允许类或结构的实例按照与数组相同的方式进行索引。索引器类似于[属性](#)，不同之处在于它们的访问器采用参数。

在下面的示例中，定义了一个泛型类，并为其提供了简单的 `get` 和 `set` 访问器方法（作为分配和检索值的方法）。`Program` 类为存储字符串创建了此类的一个实例。

C#

```
class SampleCollection<T>
{
    private T[] arr = new T[100];
    public T this[int i]
    {
        get
        {
            return arr[i];
        }
        set
        {
            arr[i] = value;
        }
    }

    // This class shows how client code uses the indexer
    class Program
    {
        static void Main(string[] args)
        {
            SampleCollection<string> stringCollection = new SampleCollection<string>();
            stringCollection[0] = "Hello, World";
            System.Console.WriteLine(stringCollection[0]);
        }
    }
}
```

索引器概述

- 索引器使得对象可按照与数组相似的方法进行索引。
- `get` 访问器返回值。`set` 访问器分配值。
- `this` 关键字用于定义索引器。
- `value` 关键字用于定义由 `set` 索引器分配的值。
- 索引器不必根据整数值进行索引，由您决定如何定义特定的查找机制。
- 索引器可被重载。
- 索引器可以有多个形参，例如当访问二维数组时。

相关章节

- [使用索引器 \(C# 编程指南\)](#)
- [接口中的索引器 \(C# 编程指南\)](#)
- [属性和索引器之间的比较 \(C# 编程指南\)](#)
- [非对称访问器可访问性 \(C# 编程指南\)](#)
- [“索引器”示例](#)
- [“索引属性”示例](#)

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.3 索引器
- 10.2.7.3 为索引器保留的成员名称
- 10.8 索引器
- 13.2.4 接口索引器

请参见

参考

[属性 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

使用索引器 (C# 编程指南)

索引器允许您按照处理数组的方式索引类、结构或接口。有关对接口使用索引器的更多信息，请参见[接口索引器](#)。

要声明类或结构上的索引器，请使用 `this` 关键字，如下例所示：

```
public int this[int index]    // Indexer declaration
{
    // get and set accessors
}
```

备注

索引器类型及其参数类型必须至少如同索引器本身一样是可访问的。有关可访问级别的更多信息，请参见[访问修饰符](#)。

索引器的签名由其形参的数量和类型组成。它不包括索引器类型或形参名。如果在同一类中声明一个以上的索引器，则它们必须具有不同的签名。

索引器值不归类为变量；因此，不能将索引器值作为 `ref` 或 `out` 参数来传递。

要为索引器提供一个其他语言可以使用的名称，请使用声明中的 `name` 属性。例如：

```
[System.Runtime.CompilerServices.IndexerName("TheItem")]
public int this [int index]    // Indexer declaration
{}
```

此索引器将具有名称 `TheItem`。不提供名称属性将生成 `Item` 默认名称。

示例 1

下面的示例说明如何声明私有数组字段、`arr` 和索引器。使用索引器可直接访问实例 `test[i]`。另一种使用索引器的方法是将数组声明为 `public` 成员并直接访问它的成员 `arr[i]`。

C#

```
class IndexerClass
{
    private int[] arr = new int[100];
    public int this[int index]    // Indexer declaration
    {
        get
        {
            // Check the index limits.
            if (index < 0 || index >= 100)
            {
                return 0;
            }
            else
            {
                return arr[index];
            }
        }
        set
        {
            if (!(index < 0 || index >= 100))
            {
                arr[index] = value;
            }
        }
    }
}

class MainClass
{
    static void Main()
```

```

    {
        IndexerClass test = new IndexerClass();
        // Call the indexer to initialize the elements #3 and #5.
        test[3] = 256;
        test[5] = 1024;
        for (int i = 0; i <= 10; i++)
        {
            System.Console.WriteLine("Element #{0} = {1}", i, test[i]);
        }
    }
}

```

输出

```

Element #0 = 0
Element #1 = 0
Element #2 = 0
Element #3 = 256
Element #4 = 0
Element #5 = 1024
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0

```

请注意，当计算索引器的访问时（例如，在 **Console.WriteLine** 语句中），将调用 **get** 访问器。因此，如果 **get** 访问器不存在，将发生编译时错误。

使用其他值进行索引

C# 并不将索引类型限制为整数。例如，对索引器使用字符串可能是有用的。通过搜索集合内的字符串并返回相应的值，可以实现此类的索引器。由于访问器可被重载，字符串和整数版本可以共存。

示例 2

在此例中，声明了存储星期几的类。声明了一个 **get** 访问器，它接受字符串（天名称），并返回相应的整数。例如，星期日将返回 0，星期一将返回 1，等等。

C#

```

// Using a string as an indexer value
class DayCollection
{
    string[] days = { "Sun", "Mon", "Tues", "Wed", "Thurs", "Fri", "Sat" };

    // This method finds the day or returns -1
    private int GetDay(string testDay)
    {
        int i = 0;
        foreach (string day in days)
        {
            if (day == testDay)
            {
                return i;
            }
            i++;
        }
        return -1;
    }

    // The get accessor returns an integer for a given string
    public int this[string day]
    {

```

```
        get
    {
        return (GetDay(day));
    }
}

class Program
{
    static void Main(string[] args)
    {
        DayCollection week = new DayCollection();
        System.Console.WriteLine(week["Fri"]);
        System.Console.WriteLine(week["Made-up Day"]);
    }
}
```

输出

5

-1

可靠编程

提高索引器的安全性和可靠性有两种主要的方法：

- 当设置并检索来自索引器访问的任何缓冲区或数组的值时，请始终确保您的代码执行范围和类型检查。
- 应当为 **get** 和 **set** 访问器的可访问性设置尽可能多的限制。这一点对 **set** 访问器尤为重要。有关更多信息，请参见[非对称访问器可访问性\(C# 编程指南\)](#)。

请参见

任务

[“索引器”示例](#)

参考

[索引器\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

接口中的索引器(C# 编程指南)

索引器可在[接口\(C# 参考\)](#)上声明。接口索引器的访问器与[类索引器](#)的访问器具有以下方面的不同：

- 接口访问器不使用修饰符。
- 接口访问器没有体。

因此，访问器的用途是指示索引器是读写、只读还是只写。

以下是接口索引器访问器的示例：

C#

```
public interface ISomeInterface
{
    //...

    // Indexer declaration:
    string this[int index]
    {
        get;
        set;
    }
}
```

一个索引器的签名必须区别于在同一接口中声明的其他所有索引器的签名。

示例

下面的示例显示如何实现接口索引器。

C#

```
// Indexer on an interface:
public interface ISomeInterface
{
    // Indexer declaration:
    int this[int index]
    {
        get;
        set;
    }
}

// Implementing the interface.
class IndexerClass : ISomeInterface
{
    private int[] arr = new int[100];
    public int this[int index]    // indexer declaration
    {
        get
        {
            // Check the index limits.
            if (index < 0 || index >= 100)
            {
                return 0;
            }
            else
            {
                return arr[index];
            }
        }
        set
        {
            if (!(index < 0 || index >= 100))

```

```

        {
            arr[index] = value;
        }
    }
}

class MainClass
{
    static void Main()
    {
        IndexerClass test = new IndexerClass();
        // Call the indexer to initialize the elements #2 and #5.
        test[2] = 4;
        test[5] = 32;
        for (int i = 0; i <= 10; i++)
        {
            System.Console.WriteLine("Element #{0} = {1}", i, test[i]);
        }
    }
}

```

输出

```

Element #0 = 0
Element #1 = 0
Element #2 = 4
Element #3 = 0
Element #4 = 0
Element #5 = 32
Element #6 = 0
Element #7 = 0
Element #8 = 0
Element #9 = 0
Element #10 = 0

```

在上例中，可以通过使用接口成员的完全限定名来使用显式接口成员实现。例如：

```

public string ISomeInterface.this
{
}

```

但是，只有当类使用同一索引器签名实现一个以上的接口时，为避免多义性才需要使用完全限定名。例如，如果 Employee 类实现的是两个接口 ICitizen 和 IEmployee，并且这两个接口具有相同的索引器签名，则必须使用显式接口成员实现。即，以下索引器声明：

```

public string IEmployee.this
{
}

```

在 IEmployee 接口上实现索引器，而以下声明：

```

public string ICitizen.this
{
}

```

在 ICitizen 接口上实现索引器。

请参见

任务

[“索引器”示例](#)

参考

[索引器\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

[接口 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

属性和索引器之间的比较(C# 编程指南)

索引器与属性类似。除下表中显示的差别外，为属性访问器定义的所有规则同样适用于索引器访问器。

属性	索引器
允许调用方法，如同它们是公共数据成员。	允许调用对象上的方法，如同对象是一个数组。
可通过简单的名称进行访问。	可通过索引器进行访问。
可以为静态成员或实例成员。	必须为实例成员。
属性的 <code>get</code> 访问器没有参数。	索引器的 <code>get</code> 访问器具有与索引器相同的形参表。
属性的 <code>set</code> 访问器包含隐式 <code>value</code> 参数。	除了 <code>value</code> 参数外，索引器的 <code>set</code> 访问器还具有与索引器相同的形参表。

请参见

参考

[索引器\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

委托 (C# 编程指南)

[委托](#)是一种引用方法的类型。一旦为委托分配了方法，委托将与该方法具有完全相同的行为。委托方法的使用可以像其他任何方法一样，具有参数和返回值，如下面的示例所示：

C#

```
public delegate int PerformCalculation(int x, int y);
```

与委托的签名（由返回类型和参数组成）匹配的任何方法都可以分配给该委托。这样就可以通过编程方式来更改方法调用，还可以向现有类中插入新代码。只要知道委托的签名，便可以分配自己的委托方法。

将方法作为参数进行引用的能力使委托成为定义回调方法的理想选择。例如，可以向排序算法传递对比较两个对象的方法的引用。分离比较代码使得可以采用更通用的方式编写算法。

委托概述

委托具有以下特点：

- 委托类似于 C++ 函数指针，但它是类型安全的。
- 委托允许将方法作为参数进行传递。
- 委托可用于定义回调方法。
- 委托可以链接在一起；例如，可以对一个事件调用多个方法。
- 方法不需要与委托签名精确匹配。有关更多信息，请参见[协变和逆变](#)。
- C# 2.0 版引入了[匿名方法](#)的概念，此类方法允许将代码块作为参数传递，以代替单独定义的方法。

本节内容

- [委托概述](#)
- [何时使用委托而不使用接口](#)
- [命名方法](#)
- [匿名方法](#)
- [协变和逆变](#)
- [如何：合并委托](#)
- [如何：声明、实例化和使用委托](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.11 委托
- 4.2.6 委托类型
- 7.5.5.2 委托调用
- 15 委托

请参见

参考

[Delegate](#)

概念

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

使用委托(C# 编程指南)

委托是一种安全地封装方法的类型，它与 C 和 C++ 中的函数指针类似。与 C 中的函数指针不同，委托是面向对象的、类型安全的和保险的。委托的类型由委托的名称定义。下面的示例声明了一个名为 `Del` 的委托，该委托可以封装一个采用字符串作为参数并返回 `void` 的方法。

C#

```
public delegate void Del(string message);
```

构造委托对象时，通常提供委托将包装的方法的名称或使用[匿名方法](#)。实例化委托后，委托将把对它进行的方法调用传递给方法。调用方传递给委托的参数被传递给方法，来自方法的返回值(如果有)由委托返回给调用方。这被称为调用委托。可以将一个实例化的委托视为被包装的方法本身来调用该委托。例如：

C#

```
// Create a method for a delegate.
public static void DelegateMethod(string message)
{
    System.Console.WriteLine(message);
}
```

C#

```
// Instantiate the delegate.
Del handler = DelegateMethod;

// Call the delegate.
handler("Hello World");
```

委托类型派生自 .NET Framework 中的 `Delegate` 类。委托类型是密封的，不能从 `Delegate` 中派生委托类型，也不可能从中派生自定义类。由于实例化委托是一个对象，所以可以将其作为参数进行传递，也可以将其赋值给属性。这样，方法便可以将一个委托作为参数来接受，并且以后可以调用该委托。这称为异步回调，是在较长的进程完成后用来通知调用方的常用方法。以这种方式使用委托时，使用委托的代码无需了解有关所用方法的实现方面的任何信息。此功能类似于接口所提供的封装。有关更多信息，请参见[何时使用委托而不使用接口](#)。

回调的另一个常见用法是定义自定义的比较方法并将该委托传递给排序方法。它允许调用方的代码成为排序算法的一部分。下面的示例方法使用 `Del` 类型作为参数：

C#

```
public void MethodWithCallback(int param1, int param2, Del callback)
{
    callback("The number is: " + (param1 + param2).ToString());
}
```

然后可以将上面创建的委托传递给该方法：

C#

```
MethodWithCallback(1, 2, handler);
```

在控制台中将收到下面的输出：

The number is: 3

在将委托用作抽象概念时，`MethodWithCallback` 不需要直接调用控制台 -- 设计它时无需考虑控制台。`MethodWithCallback` 的作用只是准备字符串并将该字符串传递给其他方法。此功能特别强大，因为委托的方法可以使用任意数量的参数。

将委托构造为包装实例方法时，该委托将同时引用实例和方法。除了它所包装的方法外，委托不了解实例类型，所以只要任意类

型的对象中具有与委托签名相匹配的方法，委托就可以引用该对象。将委托构造为包装静态方法时，它只引用方法。考虑下列声明：

C#

```
public class MethodClass
{
    public void Method1(string message) { }
    public void Method2(string message) { }
}
```

加上前面显示的静态 DelegateMethod，现在我们有三个方法可由 Del 实例进行包装。

调用委托时，它可以调用多个方法。这称为多路广播。若要向委托的方法列表（调用列表）中添加额外的方法，只需使用加法运算符或加法赋值运算符（“+”或“+=”）添加两个委托。例如：

C#

```
MethodClass obj = new MethodClass();
Del d1 = obj.Method1;
Del d2 = obj.Method2;
Del d3 = DelegateMethod;

//Both types of assignment are valid.
Del allMethodsDelegate = d1 + d2;
allMethodsDelegate += d3;
```

此时，allMethodsDelegate 在其调用列表中包含三个方法 -- Method1、Method2 和 DelegateMethod。原来的三个委托 d1、d2 和 d3 保持不变。调用 allMethodsDelegate 时，将按顺序调用所有这三个方法。如果委托使用引用参数，则引用将依次传递给三个方法中的每个方法，由一个方法引起的更改对下一个方法是可见的。如果任一方法引发了异常，而在该方法内未捕获该异常，则该异常将传递给委托的调用方，并且不再对调用列表中后面的方法进行调用。如果委托具有返回值和/或输出参数，它将返回最后调用的方法的返回值和参数。若要从调用列表中移除方法，请使用减法运算符或减法赋值运算符（“-”或“-=”）。例如：

C#

```
//remove Method1
allMethodsDelegate -= d1;

// copy AllMethodsDelegate while removing d2
Del oneMethodDelegate = allMethodsDelegate - d2;
```

由于委托类型派生自 **System.Delegate**，所以可在委托上调用该类定义的方法和属性。例如，为了找出委托的调用列表中的方法数，您可以编写下面的代码：

C#

```
int invocationCount = d1.GetInvocationList().GetLength(0);
```

在调用列表中具有多个方法的委托派生自 **MulticastDelegate**，这是 **System.Delegate** 的子类。由于两个类都支持 **GetInvocationList**，所以上面的代码在两种情况下都适用。

多路广播委托广泛用于事件处理中。事件源对象向已注册接收该事件的接收方对象发送事件通知。为了为事件注册，接收方创建了旨在处理事件的方法，然后为该方法创建委托并将该委托传递给事件源。事件发生时，源将调用委托。然后，委托调用接收方的事件处理方法并传送事件数据。给定事件的委托类型由事件源定义。有关更多信息，请参见 [事件\(C# 编程指南\)](#)。

在编译时，对分配了两种不同类型的委托进行比较将产生编译错误。如果委托实例静态地属于类型 **System.Delegate**，则允许进行比较，但在运行时将返回 false。例如：

C#

```
delegate void Delegate1();
delegate void Delegate2();

static void method(Delegate1 d, Delegate2 e, System.Delegate f)
```

```
{  
    // Compile-time error.  
    //Console.WriteLine(d == e);  
  
    // OK at compile-time. False if the run-time type of f  
    //is not the same as that of d.  
    System.Console.WriteLine(d == f);  
}
```

请参见

参考

[委托中的协变和逆变\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[委托\(C# 编程指南\)](#)

[事件\(C# 编程指南\)](#)

命名方法 (C# 编程指南)

委托可以与命名方法关联。使用命名的方法对委托进行实例化时，该方法将作为参数传递，例如：

C#

```
// Declare a delegate:  
delegate void Del(int x);  
  
// Define a named method:  
void DoWork(int k) { /* ... */ }  
  
// Instantiate the delegate using the method as a parameter:  
Del d = obj.DoWork;
```

这被称为使用命名的方法。使用命名方法构造的委托可以封装静态方法或实例方法。在以前的 C# 版本中，使用命名的方法是对委托进行实例化的唯一方式。但是，在创建新方法的系统开销不必要时，C# 2.0 允许您对委托进行实例化，并立即指定委托将在被调用时处理的代码块。这些被称为[匿名方法 \(C# 编程指南\)](#)。

备注

作为委托参数传递的方法必须与委托声明具有相同的签名。

委托实例可以封装静态或实例方法。

尽管委托可以使用 `out` 参数，但不推荐将其用于多路广播事件委托中，因为无法确定要调用哪个委托。

示例 1

以下是声明及使用委托的一个简单示例。注意，委托 `Del` 和关联的方法 `MultiplyNumbers` 具有相同的签名

C#

```
// Declare a delegate  
delegate void Del(int i, double j);  
  
class MathClass  
{  
    static void Main()  
    {  
        MathClass m = new MathClass();  
  
        // Delegate instantiation using "MultiplyNumbers"  
        Del d = m.MultiplyNumbers;  
  
        // Invoke the delegate object.  
        System.Console.WriteLine("Invoking the delegate using 'MultiplyNumbers':");  
        for (int i = 1; i <= 5; i++)  
        {  
            d(i, 2);  
        }  
    }  
  
    // Declare the associated method.  
    void MultiplyNumbers(int m, double n)  
    {  
        System.Console.Write(m * n + " ");  
    }  
}
```

输出

```
Invoking the delegate using 'MultiplyNumbers':  
2 4 6 8 10
```

示例 2

在下面的示例中，一个委托被同时映射到静态方法和实例方法，并分别返回特定的信息。

C#

```
// Declare a delegate
delegate void Del();

class SampleClass
{
    public void InstanceMethod()
    {
        System.Console.WriteLine("A message from the instance method.");
    }

    static public void StaticMethod()
    {
        System.Console.WriteLine("A message from the static method.");
    }
}

class TestSampleClass
{
    static void Main()
    {
        SampleClass sc = new SampleClass();

        // Map the delegate to the instance method:
        Del d = sc.InstanceMethod;
        d();

        // Map to the static method:
        d = SampleClass.StaticMethod;
        d();
    }
}
```

输出

A message from the instance method.

A message from the static method.

请参见

任务

[如何：合并委托（多路广播委托）\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[委托\(C# 编程指南\)](#)

[事件\(C# 编程指南\)](#)

匿名方法(C# 编程指南)

在 2.0 之前的 C# 版本中，声明委托的唯一方法是使用**命名方法**。C# 2.0 引入了匿名方法。

要将代码块传递为委托参数，创建匿名方法则是唯一的方法。例如：

C#

```
// Create a handler for a click event
button1.Click += delegate(System.Object o, System.EventArgs e)
    { System.Windows.Forms.MessageBox.Show("Click!"); };
```

或

C#

```
// Create a delegate instance
delegate void Del(int x);

// Instantiate the delegate using an anonymous method
Del d = delegate(int k) { /* ... */ };
```

如果使用匿名方法，则不必创建单独的方法，因此减少了实例化委托所需的编码系统开销。

例如，如果创建方法所需的系统开销是不必要的，在委托的位置指定代码块就非常有用。启动新线程即是一个很好的示例。无需为委托创建更多方法，线程类即可创建一个线程并且包含该线程执行的代码。

C#

```
void StartThread()
{
    System.Threading.Thread t1 = new System.Threading.Thread
        (delegate()
        {
            System.Console.Write("Hello, ");
            System.Console.WriteLine("World!");
        });
    t1.Start();
}
```

备注

匿名方法的参数的范围是 *anonymous-method-block*。

在目标在块外部的匿名方法块内使用跳转语句(如 **goto**、**break** 或 **continue**)是错误的。在目标在块内部的匿名方法块外部使用跳转语句(如 **goto**、**break** 或 **continue**)也是错误的。

如果局部变量和参数的范围包含匿名方法声明，则该局部变量和参数称为该匿名方法的外部变量或捕获变量。例如，下面代码段中的 **n** 即是一个外部变量：

C#

```
int n = 0;
Del d = delegate() { System.Console.WriteLine("Copy #: {0}", ++n); };
```

与局部变量不同，外部变量的生命周期一直持续到引用该匿名方法的委托符合垃圾回收的条件为止。对 **n** 的引用是在创建该委托时捕获的。

匿名方法不能访问外部范围的 **ref** 或 **out** 参数。

在 *anonymous-method-block* 中不能访问任何不安全代码。

示例

下面的示例演示实例化委托的两种方法：

- 使委托与匿名方法关联。
- 使委托与命名方法 (DoWork) 关联。

两种方法都会在调用委托时显示一条消息。

C#

```
// Declare a delegate
delegate void Printer(string s);

class TestClass
{
    static void Main()
    {
        // Instantiate the delegate type using an anonymous method:
        Printer p = delegate(string j)
        {
            System.Console.WriteLine(j);
        };

        // Results from the anonymous delegate call:
        p("The delegate using the anonymous method is called.");

        // The delegate instantiation using a named method "DoWork":
        p = new Printer(TestClass.DoWork);

        // Results from the old style delegate call:
        p("The delegate using the named method is called.");
    }

    // The method associated with the named delegate:
    static void DoWork(string k)
    {
        System.Console.WriteLine(k);
    }
}
```

输出

The delegate using the anonymous method is called.

The delegate using the named method is called.

请参见

参考

[方法\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[委托\(C# 编程指南\)](#)

[不安全代码和指针\(C# 编程指南\)](#)

[命名方法\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

何时使用委托而不使用接口 (C# 编程指南)

委托和接口都允许类设计器分离类型声明和实现。给定的[接口](#)可由任何[类或结构](#)继承和实现;可以为任何类中的方法创建[委托](#),前提是该方法符合委托的方法签名。接口引用或委托可由不了解实现该接口或委托方法的类的对象使用。既然存在这些相似性,那么类设计器何时应使用委托,何时又该使用接口呢?

在以下情况中使用委托:

- 当使用事件设计模式时。
- 当封装静态方法可取时。
- 当调用方不需要访问实现该方法的对象中的其他属性、方法或接口时。
- 需要方便的组合。
- 当类可能需要该方法的多个实现时。

在以下情况中使用接口:

- 当存在一组可能被调用的相关方法时。
- 当类只需要方法的单个实现时。
- 当使用接口的类想要将该接口强制转换为其他接口或类类型时。
- 当正在实现的方法链接到类的类型或标识时:例如比较方法。

使用单一方法接口而不使用委托的一个很好的示例是[IComparable](#) 或 [IComparable](#)。[IComparable](#) 声明 [CompareTo](#) 方法,该方法返回一个整数,以指定相同类型的两个对象之间的小于、等于或大于关系。[IComparable](#) 可用作排序算法的基础,虽然将委托比较方法用作排序算法的基础是有效的,但是并不理想。因为进行比较的能力属于类,而比较算法不会在运行时改变,所以单一方法接口是理想的。

请参见

参考

[方法 \(C# 编程指南\)](#)

[接口 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

委托中的协变和逆变(C# 编程指南)

将方法签名与[委托](#)类型匹配时，协变和逆变为您提供了一定程度的灵活性。协变允许方法具有的派生返回类型比委托中定义的更多。逆变允许方法具有的派生参数类型比委托类型中的更少。

示例 1(协变)

本示例演示如何将委托与具有返回类型的方法一起使用，这些返回类型派生自委托签名中的返回类型。由 `SecondHandler` 返回的数据类型是 `Dogs` 类型，它是由委托中定义的 `Mammals` 类型派生的。

C#

```
class Mammals
{
}

class Dogs : Mammals
{
}

class Program
{
    // Define the delegate.
    public delegate Mammals HandlerMethod();

    public static Mammals FirstHandler()
    {
        return null;
    }

    public static Dogs SecondHandler()
    {
        return null;
    }

    static void Main()
    {
        HandlerMethod handler1 = FirstHandler;

        // Covariance allows this delegate.
        HandlerMethod handler2 = SecondHandler;
    }
}
```

示例 2(逆变)

本示例演示如何将委托与具有某个类型的参数的方法一起使用，这些参数是委托签名参数类型的基类型。通过逆变，以前必须使用若干个不同处理程序的地方现在只要使用一个事件处理程序即可。如，现在可以创建一个接收 `EventArgs` 输入参数的事件处理程序，然后，可以将该处理程序与发送 `MouseEventArgs` 类型(作为参数)的 `Button.MouseClick` 事件一起使用，也可以将该处理程序与发送 `KeyEventArgs` 参数的 `TextBox.KeyDown` 事件一起使用。

C#

```
System.DateTime lastActivity;
public Form1()
{
    InitializeComponent();

    lastActivity = new System.DateTime();
    this.textBox1.KeyDown += this.MultiHandler; //works with KeyEventArgs
    this.button1.MouseClick += this.MultiHandler; //works with MouseEventArgs
}

// Event hander for any event with an EventArgs or
```

```
// derived class in the second parameter
private void MultiHandler(object sender, System.EventArgs e)
{
    lastActivity = System.DateTime.Now;
}
```

请参见

参考

[泛型委托\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[事件\(C# 编程指南\)](#)

[委托\(C# 编程指南\)](#)

如何：合并委托（多路广播委托）(C# 编程指南)

本示例演示如何组合多路广播委托。[委托](#)对象的一个用途在于，可以使用 + 运算符将它们分配给一个要成为多路广播委托的委托实例。组合的委托可调用组成它的那两个委托。只有相同类型的委托才可以组合。

- 运算符可用来从组合的委托移除组件委托。

示例

C#

```

delegate void Del(string s);

class TestClass
{
    static void Hello(string s)
    {
        System.Console.WriteLine(" Hello, {0}!", s);
    }

    static void Goodbye(string s)
    {
        System.Console.WriteLine(" Goodbye, {0}!", s);
    }

    static void Main()
    {
        Del a, b, c, d;

        // Create the delegate object a that references
        // the method Hello:
        a = Hello;

        // Create the delegate object b that references
        // the method Goodbye:
        b = Goodbye;

        // The two delegates, a and b, are composed to form c:
        c = a + b;

        // Remove a from the composed delegate, leaving d,
        // which calls only the method Goodbye:
        d = c - a;

        System.Console.WriteLine("Invoking delegate a:");
        a("A");
        System.Console.WriteLine("Invoking delegate b:");
        b("B");
        System.Console.WriteLine("Invoking delegate c:");
        c("C");
        System.Console.WriteLine("Invoking delegate d:");
        d("D");
    }
}

```

输出

```

Invoking delegate a:
Hello, A!
Invoking delegate b:
Goodbye, B!
Invoking delegate c:
Hello, C!
Goodbye, C!
Invoking delegate d:
Goodbye, D!

```

[请参见](#)

[参考](#)

[MulticastDelegate](#)

[概念](#)

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

如何: 声明、实例化和使用委托 (C# 编程指南)

委托的声明如下所示：

C#

```
public delegate void Del<T>(T item);
public void Notify(int i) { }
```

C#

```
Del<int> d1 = new Del<int>(Notify);
```

在 C# 2.0 中，还可以使用下面的简化语法来声明委托：

C#

```
Del<int> d2 = Notify;
```

下面的示例阐释声明、实例化和使用委托。BookDB 类封装一个书店数据库，它维护一个书籍数据库。它公开 ProcessPaperbackBooks 方法，该方法在数据库中查找所有平装书，并对每本平装书调用一个委托。所使用的 **delegate** 类型称为 ProcessBookDelegate。Test 类使用该类输出平装书的书名和平均价格。

委托的使用促进了书店数据库和客户代码之间功能的良好分隔。客户代码不知道书籍的存储方式和书店代码查找平装书的方式。书店代码也不知道找到平装书后将对平装书进行什么处理。

示例

C#

```
// A set of classes for handling a bookstore:
namespace Bookstore
{
    using System.Collections;

    // Describes a book in the book list:
    public struct Book
    {
        public string Title;           // Title of the book.
        public string Author;          // Author of the book.
        public decimal Price;          // Price of the book.
        public bool Paperback;         // Is it paperback?

        public Book(string title, string author, decimal price, bool paperBack)
        {
            Title = title;
            Author = author;
            Price = price;
            Paperback = paperBack;
        }
    }

    // Declare a delegate type for processing a book:
    public delegate void ProcessBookDelegate(Book book);

    // Maintains a book database.
    public class BookDB
    {
        // List of all books in the database:
        ArrayList list = new ArrayList();

        // Add a book to the database:
        public void AddBook(string title, string author, decimal price, bool paperBack)
```

```

        {
            list.Add(new Book(title, author, price, paperBack));
        }

        // Call a passed-in delegate on each paperback book to process it:
        public void ProcessPaperbackBooks(ProcessBookDelegate processBook)
        {
            foreach (Book b in list)
            {
                if (b.Paperback)
                    // Calling the delegate:
                    processBook(b);
            }
        }
    }
}

// Using the Bookstore classes:
namespace BookTestClient
{
    using Bookstore;

    // Class to total and average prices of books:
    class PriceTotaller
    {
        int countBooks = 0;
        decimal priceBooks = 0.0m;

        internal void AddBookToTotal(Book book)
        {
            countBooks += 1;
            priceBooks += book.Price;
        }

        internal decimal AveragePrice()
        {
            return priceBooks / countBooks;
        }
    }

    // Class to test the book database:
    class TestBookDB
    {
        // Print the title of the book.
        static void PrintTitle(Book b)
        {
            System.Console.WriteLine(" {0}", b.Title);
        }

        // Execution starts here.
        static void Main()
        {
            BookDB bookDB = new BookDB();

            // Initialize the database with some books:
            AddBooks(bookDB);

            // Print all the titles of paperbacks:
            System.Console.WriteLine("Paperback Book Titles:");

            // Create a new delegate object associated with the static
            // method Test.PrintTitle:
            bookDB.ProcessPaperbackBooks(PrintTitle);

            // Get the average price of a paperback by using
            // a PriceTotaller object:
            PriceTotaller totaller = new PriceTotaller();
        }
    }
}

```

```

// Create a new delegate object associated with the nonstatic
// method AddBookToTotal on the object totaller:
bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);

System.Console.WriteLine("Average Paperback Book Price: ${0:#.##}",
    totaller.AveragePrice());
}

// Initialize the book database with some test books:
static void AddBooks(BookDB bookDB)
{
    bookDB.AddBook("The C Programming Language", "Brian W. Kernighan and Dennis M.
Ritchie", 19.95m, true);
    bookDB.AddBook("The Unicode Standard 2.0", "The Unicode Consortium", 39.95m, tr
ue);
    bookDB.AddBook("The MS-DOS Encyclopedia", "Ray Duncan", 129.95m, false);
    bookDB.AddBook("Dogbert's Clues for the Clueless", "Scott Adams", 12.00m, true)
;
}
}
}

```

输出

Paperback Book Titles:
The C Programming Language
The Unicode Standard 2.0
Dogbert's Clues for the Clueless
Average Paperback Book Price: \$23.97

可靠编程

- 声明委托。

下列语句：

C#

```
public delegate void ProcessBookDelegate(Book book);
```

声明一个新的委托类型。每个委托类型都描述参数的数目和类型，以及它可以封装的方法的返回值类型。每当需要一组新的参数类型或新的返回值类型时，都必须声明一个新的委托类型。

- 实例化委托。

声明了委托类型后，必须创建委托对象并使之与特定方法关联。在上面的示例中，这是通过将 `PrintTitle` 方法传递给 `ProcessPaperbackBooks` 方法来完成的，如下所示：

C#

```
bookDB.ProcessPaperbackBooks(PrintTitle);
```

这将创建与 `静态方法 Test.PrintTitle` 关联的新委托对象。类似地，对象 `totaller` 的非静态方法 `AddBookToTotal` 是按如下方式传递的：

C#

```
bookDB.ProcessPaperbackBooks(totaller.AddBookToTotal);
```

在两个示例中，都向 `ProcessPaperbackBooks` 方法传递了一个新的委托对象。

委托一旦创建，它的关联方法就不能更改；委托对象是不可变的。

- 调用委托。

创建委托对象后，通常将委托对象传递给将调用该委托的其他代码。通过委托对象的名称（后面跟着要传递给委托的参数，括在括号内）调用委托对象。下面是委托调用的示例：

C#

```
processBook(b);
```

与本例一样，可以通过使用 **BeginInvoke** 和 **EndInvoke** 方法同步或异步调用委托。

请参见

概念

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

[委托 \(C# 编程指南\)](#)

事件 (C# 编程指南)

在发生其他类或对象关注的事情时，[类](#)或对象可通过事件通知它们。发送(或引发)事件的类称为“发行者”，接收(或处理)事件的类称为“订户”。

在典型的 C# Windows 窗体或 Web 应用程序中，可订阅由控件(如按钮和列表框)引发的事件。可使用 Visual C# 集成开发环境(IDE)来浏览控件发布的事件，选择要处理的事件。IDE 会自动添加空事件处理方法和订阅事件的代码。有关更多信息，请参见[如何:订阅和取消订阅事件 \(C# 编程指南\)](#)。

事件概述

事件具有以下特点：

- 发行者确定何时引发事件，订户确定执行何种操作来响应该事件。
- 一个事件可以有多个订户。一个订户可处理来自多个发行者的多个事件。
- 没有订户的事件永远不会被调用。
- 事件通常用于通知用户操作(如：图形用户界面中的按钮单击或菜单选择操作)。
- 如果一个事件有多个订户，当引发该事件时，会同步调用多个事件处理程序。要异步调用事件，请参见[使用异步方式调用同步方法](#)。
- 可以利用事件同步线程。
- 在 .NET Framework 类库中，事件是基于 [EventHandler](#) 委托和 [EventArgs](#) 基类的。

相关章节

有关更多信息，请参见：

- [如何:订阅和取消订阅事件 \(C# 编程指南\)](#)
- [如何:发布符合 .NET Framework 准则的事件 \(C# 编程指南\)](#)
- [如何:引发派生类中的基类事件 \(C# 编程指南\)](#)
- [如何:实现接口事件 \(C# 编程指南\)](#)
- [线程同步 \(C# 编程指南\)](#)
- [如何:使用字典存储事件实例 \(C# 编程指南\)](#)
- [“事件”示例](#)
- [事件设计](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.4 事件
- 10.2.7.2 事件的保留成员名称
- 10.7 事件
- 13.2.3 接口事件

请参见

参考

[EventHandler](#)

概念

[C# 编程指南](#)

[委托 \(C# 编程指南\)](#)

其他资源

[在 Windows 窗体中创建事件处理程序](#)

如何：订阅和取消订阅事件 (C# 编程指南)

如果您想编写引发事件时调用的自定义代码，则可以订阅由其他类发布的事件。例如，可以订阅某个按钮的“单击”事件，以使应用程序在用户单击该按钮时执行一些有用的操作。

使用 Visual Studio 2005 IDE 订阅事件

- 如果“属性”窗口不可见，请在“设计”视图中，右击要创建事件处理程序的窗体或控件，然后选择“属性”。
- 在“属性”窗口的顶部，单击“事件”图标。
- 双击要创建的事件，例如 `Load` 事件。

Visual C# 会创建一个空事件处理程序方法，并将其添加到您的代码中。或者，您也可以在“代码”视图中手动添加代码。例如，下面的代码行声明了一个在 `Form` 类引发 `Load` 事件时调用的事件处理程序方法。

C#

```
private void Form1_Load(object sender, System.EventArgs e)
{
    // Add your form load event handling code here.
}
```

订阅该事件所需的代码行也会在您项目的 `Form1.Designer.cs` 文件的 `InitializeComponent` 方法中自动生成。该代码行类似于：

```
this.Load += new System.EventHandler(this.Form1_Load);
```

以编程方式订阅事件

- 定义一个事件处理程序方法，其签名与该事件的委托签名匹配。例如，如果事件基于 `EventHandler` 委托类型，则下面的代码表示方法存根：

```
void HandleCustomEvent(object sender, CustomEventArgs a)
{
    // Do something useful here.
}
```

- 使用加法赋值运算符 (`+=`) 来为事件附加事件处理程序。在下面的示例中，假设名为 `publisher` 的对象拥有一个名为 `RaiseCustomEvent` 的事件。请注意，订户类需要引用发行者类才能订阅其事件。

```
publisher.RaiseCustomEvent += HandleCustomEvent;
```

请注意，上面的语法是 C# 2.0 中的新语法。它完全等效于 C# 1.0 语法，必须使用以下新关键字显式创建封装委托：

```
publisher.RaiseCustomEvent += new CustomEventHandler(HandleCustomEvent);
```

使用匿名方法订阅事件

- 使用加法赋值运算符 (`+=`) 来为事件附加匿名方法。在下面的示例中，假设名为 `publisher` 的对象拥有一个名为 `RaiseCustomEvent` 的事件，并且还定义了一个 `CustomEventArgs` 类以承载某些类型的专用事件信息。请注意，订户类需要引用 `publisher` 才能订阅其事件。

```
publisher.RaiseCustomEvent += delegate(object o, CustomEventArgs e)
{
```

```
        string s = o.ToString() + " " + e.ToString();
        Console.WriteLine(s);
    };
}
```

请注意，如果您是使用匿名方法订阅的事件，该事件的取消订阅过程就比较麻烦。此时要取消订阅，请返回到该事件的订阅代码，将该匿名方法存储在委托变量中，然后将委托添加到该事件中。

取消订阅

要防止在引发事件时调用事件处理程序，您只需取消订阅该事件。要防止资源泄露，请在释放订户对象之前取消订阅事件，这一点很重要。在取消订阅事件之前，在发布对象中作为该事件的基础的多路广播委托会引用封装了订户的事件处理程序的委托。只要发布对象包含该引用，就不会对订户对象执行垃圾回收。

取消订阅事件

- 使用减法赋值运算符 (-=) 取消订阅事件：

```
publisher.RaiseCustomEvent -= HandleCustomEvent;
```

所有订户都取消订阅某事件后，发行者类中的事件实例会设置为 null。

请参见

任务

[如何：发布符合 .NET Framework 准则的事件 \(C# 编程指南\)](#)

参考

[event \(C# 参考\)](#)

[-= 运算符 \(C# 参考\)](#)

[+= 运算符 \(C# 参考\)](#)

概念

[事件 \(C# 编程指南\)](#)

如何：发布符合 .NET Framework 准则的事件 (C# 编程指南)

下面的过程演示了如何将符合标准 .NET Framework 模式的事件添加到您自己的类和结构中。.NET Framework 类库中的所有事件均基于 [EventHandler](#) 委托，定义如下：

```
public delegate void EventHandler(object sender, EventArgs e);
```

注意

.NET Framework 2.0 引入了此委托的一个泛型版本，即 [EventHandler<T>](#)。下面的示例显示如何使用这两种版本。

虽然您定义的类中的事件可采用任何有效委托类型(包括会返回值的委托)，但是，通常建议您使用 **EventHandler** 让事件采用 .NET Framework 模式，如下面的示例所示：

采用 **EventHandler** 模式发布事件

1. (如果不需要发送含事件的自定义数据，请跳过此步骤，直接进入步骤 3a。)在发行者类和订户类均可看见的范围中声明类，并添加保留自定义事件数据所需的成员。在此示例中，会返回一个简单字符串。

```
public class CustomEventArgs : EventArgs
{
    public CustomEventArgs(string s)
    {
        msg = s;
    }
    private string msg;
    public string Message
    {
        get { return msg; }
    }
}
```

2. (如果您使用的是 **EventHandler** 的泛型版本，请跳过此步骤。)在发布类中声明一个委托。为它指定以 **EventHandler** 结尾的名称。第二个参数指定自定义 **EventArgs** 类型。

```
public delegate void CustomEventHandler(object sender, CustomEventArgs a);
```

3. 使用以下任一步骤，在发布类中声明事件。

- a. 如果没有自定义 **EventArgs** 类，事件类型就是非泛型 **EventHandler** 委托。它无需声明，因为它已在 C# 项目默认包含的 [System](#) 命名空间中进行了声明：

```
public event EventHandler RaiseCustomEvent;
```

- b. 如果使用的是 **EventHandler** 的非泛型版本，并且您有一个由 [EventArgs](#) 派生的自定义类，请在发布类中声明您的事件，并且将您的委托用作类型：

```
class Publisher
{
    public event CustomEventHandler RaiseCustomEvent;
}
```

- c. 如果使用的是泛型版本，则不需要自定义委托。相反，应将事件类型指定为 **EventHandler<CustomEventArgs>**，在尖括号内放置您自己的类的名称。

```
public event EventHandler<CustomEventArgs> RaiseCustomEvent;
```

示例

下面的示例演示了上述步骤，它将自定义 EventArgs 类和 EventHandler<T> 用作事件类型。

C#

```
namespace DotNetEvents
{
    using System;
    using System.Collections.Generic;

    // Define a class to hold custom event info
    public class CustomEventArgs : EventArgs
    {
        public CustomEventArgs(string s)
        {
            message = s;
        }
        private string message;

        public string Message
        {
            get { return message; }
            set { message = value; }
        }
    }

    // Class that publishes an event
    class Publisher
    {

        // Declare the event using EventHandler<T>
        public event EventHandler<CustomEventArgs> RaiseCustomEvent;

        public void DoSomething()
        {
            // Write some code that does something useful here
            // then raise the event. You can also raise an event
            // before you execute a block of code.
            OnRaiseCustomEvent(new CustomEventArgs("Did something"));

        }

        // Wrap event invocations inside a protected virtual method
        // to allow derived classes to override the event invocation behavior
        protected virtual void OnRaiseCustomEvent(CustomEventArgs e)
        {
            // Make a temporary copy of the event to avoid possibility of
            // a race condition if the last subscriber unsubscribes
            // immediately after the null check and before the event is raised.
            EventHandler<CustomEventArgs> handler = RaiseCustomEvent;

            // Event will be null if there are no subscribers
            if (handler != null)
            {
                // Format the string to send inside the CustomEventArgs parameter
                e.Message += String.Format(" at {0}", DateTime.Now.ToString());

                // Use the () operator to raise the event.
                handler(this, e);
            }
        }
    }

    //Class that subscribes to an event
    class Subscriber
```

```
{  
    private string id;  
    public Subscriber(string ID, Publisher pub)  
    {  
        id = ID;  
        // Subscribe to the event using C# 2.0 syntax  
        pub.RaiseCustomEvent += HandleCustomEvent;  
    }  
  
    // Define what actions to take when the event is raised.  
    void HandleCustomEvent(object sender, CustomEventArgs e)  
    {  
        Console.WriteLine(id + " received this message: {0}", e.Message);  
    }  
}  
  
class Program  
{  
    static void Main(string[] args)  
    {  
        Publisher pub = new Publisher();  
        Subscriber sub1 = new Subscriber("sub1", pub);  
        Subscriber sub2 = new Subscriber("sub2", pub);  
  
        // Call the method that raises the event.  
        pub.DoSomething();  
  
        // Keep the console window open  
        Console.WriteLine("Press Enter to close this window.");  
        Console.ReadLine();  
    }  
}
```

请参见

参考

[Delegate](#)

概念

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

[事件设计](#)

[委托 \(C# 编程指南\)](#)

如何：引发派生类中的基类事件 (C# 编程指南)

以下简单示例演示了在基类中声明可从派生类引发的事件的标准方法。此模式广泛应用于 .NET Framework 基类库中的 Windows 窗体类。

在创建可用作其他类的基类的类时，必须考虑如下事实：事件是特殊类型的委托，只可以从声明它们的类中调用。派生类无法直接调用基类中声明的事件。尽管有时您可能希望某个事件只能通过基类引发，但在大多数情形下，您应该允许派生类调用基类事件。为此，您可以在包含该事件的基类中创建一个受保护的调用方法。通过调用或重写此调用方法，派生类便可以间接调用该事件。

示例

C#

```

namespace BaseClassEvents
{
    using System;
    using System.Collections.Generic;

    // Special EventArgs class to hold info about Shapes.
    public class ShapeEventArgs : EventArgs
    {
        private double newArea;

        public ShapeEventArgs(double d)
        {
            newArea = d;
        }
        public double NewArea
        {
            get { return newArea; }
        }
    }

    // Base class event publisher
    public abstract class Shape
    {
        protected double area;

        public double Area
        {
            get { return area; }
            set { area = value; }
        }
        // The event. Note that by using the generic EventHandler<T> event type
        // we do not need to declare a separate delegate type.
        public event EventHandler<ShapeEventArgs> ShapeChanged;

        public abstract void Draw();

        //The event-invoking method that derived classes can override.
        protected virtual void OnShapeChanged(ShapeEventArgs e)
        {
            // Make a temporary copy of the event to avoid possibility of
            // a race condition if the last subscriber unsubscribes
            // immediately after the null check and before the event is raised.
            EventHandler<ShapeEventArgs> handler = ShapeChanged;
            if (handler != null)
            {
                handler(this, e);
            }
        }
    }

    public class Circle : Shape
    {

```

```

private double radius;
public Circle(double d)
{
    radius = d;
    area = 3.14 * radius;
}
public void Update(double d)
{
    radius = d;
    area = 3.14 * radius;
    OnShapeChanged(new ShapeEventArgs(area));
}
protected override void OnShapeChanged(ShapeEventArgs e)
{
    // Do any circle-specific processing here.

    // Call the base class event invocation method.
    base.OnShapeChanged(e);
}
public override void Draw()
{
    Console.WriteLine("Drawing a circle");
}
}

public class Rectangle : Shape
{
private double length;
private double width;
public Rectangle(double length, double width)
{
    this.length = length;
    this.width = width;
    area = length * width;
}
public void Update(double length, double width)
{
    this.length = length;
    this.width = width;
    area = length * width;
    OnShapeChanged(new ShapeEventArgs(area));
}
protected override void OnShapeChanged(ShapeEventArgs e)
{
    // Do any rectangle-specific processing here.

    // Call the base class event invocation method.
    base.OnShapeChanged(e);
}
public override void Draw()
{
    Console.WriteLine("Drawing a rectangle");
}
}

// Represents the surface on which the shapes are drawn
// Subscribes to shape events so that it knows
// when to redraw a shape.
public class ShapeContainer
{
List<Shape> _list;

public ShapeContainer()
{
    _list = new List<Shape>();
}

```

```

public void AddShape(Shape s)
{
    _list.Add(s);
    // Subscribe to the base class event.
    s.ShapeChanged += HandleShapeChanged;
}

// ...Other methods to draw, resize, etc.

private void HandleShapeChanged(object sender, ShapeEventArgs e)
{
    Shape s = (Shape)sender;

    // Diagnostic message for demonstration purposes.
    Console.WriteLine("Received event. Shape area is now {0}", e.NewArea);

    // Redraw the shape here.
    s.Draw();
}
}

class Test
{

    static void Main(string[] args)
    {
        //Create the event publishers and subscriber
        Circle c1 = new Circle(54);
        Rectangle r1 = new Rectangle(12, 9);
        ShapeContainer sc = new ShapeContainer();

        // Add the shapes to the container.
        sc.AddShape(c1);
        sc.AddShape(r1);

        // Cause some events to be raised.
        c1.Update(57);
        r1.Update(7, 7);

        // Keep the console window open.
        Console.WriteLine();
        Console.WriteLine("Press Enter to exit");
        Console.ReadLine();
    }
}
}

```

输出

```

Received event. Shape area is now 178.98
Drawing a circle
Received event. Shape area is now 49
Drawing a rectangle

```

请参见

参考

[访问修饰符 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[事件 \(C# 编程指南\)](#)

[委托 \(C# 编程指南\)](#)

其他资源

[在 Windows 窗体中创建事件处理程序](#)

如何：实现接口事件 (C# 编程指南)

[接口](#)可声明事件。下面的示例演示如何在类中实现接口事件。接口事件的实现规则与任何接口方法或属性的实现规则基本相同。

在类中实现接口事件

- 在类中声明事件，然后在适当的位置调用该事件。

```
public interface IDrawingObject
{
    event EventHandler ShapeChanged;
}

public class MyEventArgs : EventArgs {...}

public class Shape : IDrawingObject
{
    event EventHandler ShapeChanged;
    void ChangeShape()
    {
        // Do something before the event...
        OnShapeChanged(new MyEventArgs(...));
        // or do something after the event.
    }
    protected virtual void OnShapeChanged(MyEventArgs e)
    {
        if(ShapeChanged != null)
        {
            ShapeChanged(this, e);
        }
    }
}
```

示例

下面的示例演示如何处理以下的不常见情况：您的类是从两个以上的接口继承的，每个接口都含有同名事件）。在这种情况下，您至少要为其中一个事件提供显式接口实现。为事件编写显式接口实现时，必须编写 **add** 和 **remove** 事件访问器。这两个事件访问器通常由编译器提供，但在这种情况下编译器不能提供。

您可以提供自己的访问器，以便指定这两个事件是由您的类中的同一事件表示，还是由不同事件表示。例如，根据接口规范，如果事件应在不同时间引发，则可以将每个事件与类中的一个单独实现关联。在下面的示例中，订户将形状引用强制转换为 `IShape` 或 `IDrawingObject`，从而确定自己将会接收哪个 `OnDraw` 事件。

C#

```
namespace WrapTwoInterfaceEvents
{
    using System;

    public interface IDrawingObject
    {
        // Raise this event before drawing
        // the object.
        event EventHandler OnDraw;
    }

    public interface IShape
    {
        // Raise this event after drawing
        // the shape.
        event EventHandler OnDraw;
    }
}
```

```

// Base class event publisher inherits two
// interfaces, each with an OnDraw event
public class Shape : IDrawingObject, IShape
{
    // Create an event for each interface event
    event EventHandler PreDrawEvent;
    event EventHandler PostDrawEvent;

    // Explicit interface implementation required.
    // Associate IDrawingObject's event with
    // PreDrawEvent
    event EventHandler IDrawingObject.OnDraw
    {
        add { PreDrawEvent += value; }
        remove { PreDrawEvent -= value; }
    }
    // Explicit interface implementation required.
    // Associate IShape's event with
    // PostDrawEvent
    event EventHandler IShape.OnDraw
    {
        add { PostDrawEvent += value; }
        remove { PostDrawEvent -= value; }
    }

    // For the sake of simplicity this one method
    // implements both interfaces.
    public void Draw()
    {
        // Raise IDrawingObject's event before the object is drawn.
        EventHandler handler = PreDrawEvent;
        if (handler != null)
        {
            handler(this, new EventArgs());
        }
        Console.WriteLine("Drawing a shape.");

        // RaiseIShape's event after the object is drawn.
        handler = PostDrawEvent;
        if (handler != null)
        {
            handler(this, new EventArgs());
        }
    }
}
public class Subscriber1
{
    // References the shape object as an IDrawingObject
    public Subscriber1(Shape shape)
    {
        IDrawingObject d = (IDrawingObject)shape;
        d.OnDraw += new EventHandler(d_OnDraw);
    }

    void d_OnDraw(object sender, EventArgs e)
    {
        Console.WriteLine("Sub1 receives the IDrawingObject event.");
    }
}
// References the shape object as an IShape
public class Subscriber2
{
    public Subscriber2(Shape shape)
    {
        IShape d = (IShape)shape;
        d.OnDraw += new EventHandler(d_OnDraw);
    }
}

```

```
void d_OnDraw(object sender, EventArgs e)
{
    Console.WriteLine("Sub2 receives the IShape event.");
}
}

public class Program
{
    static void Main(string[] args)
    {
        Shape shape = new Shape();
        Subscriber1 sub = new Subscriber1(shape);
        Subscriber2 sub2 = new Subscriber2(shape);
        shape.Draw();

        Console.WriteLine("Press Enter to close this window.");
        Console.ReadLine();
    }
}
```

输出

```
Sub1 receives the IDrawingObject event.
Drawing a shape.
Sub2 receives the IShape event.
```

请参见

任务

[如何:引发派生类中的基类事件\(C# 编程指南\)](#)

参考

[显式接口实现\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[事件\(C# 编程指南\)](#)

[委托\(C# 编程指南\)](#)

如何：使用字典存储事件实例(C# 编程指南)

accessor-declarations 的一种用法是公开大量的事件但不为每个事件分配字段，而是使用字典来存储这些事件实例。这只有在具有非常多的事件、但您预计大部分事件都不会实现时才有用。

示例

C#

```

public delegate void EventHandler1(int i);
public delegate void EventHandler2(string s);

public class PropertyEventsSample
{
    private System.Collections.Generic.Dictionary<string, System.Delegate> eventTable;

    public PropertyEventsSample()
    {
        eventTable = new System.Collections.Generic.Dictionary<string, System.Delegate>();
        eventTable.Add("Event1", null);
        eventTable.Add("Event2", null);
    }

    public event EventHandler1 Event1
    {
        add
        {
            eventTable["Event1"] = (EventHandler1)eventTable["Event1"] + value;
        }
        remove
        {
            eventTable["Event1"] = (EventHandler1)eventTable["Event1"] - value;
        }
    }

    public event EventHandler2 Event2
    {
        add
        {
            eventTable["Event2"] = (EventHandler2)eventTable["Event2"] + value;
        }
        remove
        {
            eventTable["Event2"] = (EventHandler2)eventTable["Event2"] - value;
        }
    }

    internal void RaiseEvent1(int i)
    {
        EventHandler1 handler1;
        if (null != (handler1 = (EventHandler1)eventTable["Event1"]))
        {
            handler1(i);
        }
    }

    internal void RaiseEvent2(string s)
    {
        EventHandler2 handler2;
        if (null != (handler2 = (EventHandler2)eventTable["Event2"]))
        {
            handler2(s);
        }
    }
}

public class TestClass

```

```
{  
    public static void Delegate1Method(int i)  
    {  
        System.Console.WriteLine(i);  
    }  
  
    public static void Delegate2Method(string s)  
    {  
        System.Console.WriteLine(s);  
    }  
  
    static void Main()  
    {  
        PropertyEventsSample p = new PropertyEventsSample();  
  
        p.Event1 += new EventHandler1(TestClass.Delegate1Method);  
        p.Event1 += new EventHandler1(TestClass.Delegate1Method);  
        p.Event1 -= new EventHandler1(TestClass.Delegate1Method);  
        p.RaiseEvent1(2);  
  
        p.Event2 += new EventHandler2(TestClass.Delegate2Method);  
        p.Event2 += new EventHandler2(TestClass.Delegate2Method);  
        p.Event2 -= new EventHandler2(TestClass.Delegate2Method);  
        p.RaiseEvent2("TestString");  
    }  
}
```

输出

```
2  
TestString
```

请参见

概念

[C# 编程指南](#)

[事件\(C# 编程指南\)](#)

[委托\(C# 编程指南\)](#)

使用基于事件的异步模式进行多线程编程

有多种方式可向客户端代码公开异步功能。基于事件的异步模式为类规定了用于显示异步行为的建议方式。

本节内容

[基于事件的异步模式概述](#)

描述基于事件的异步模式如何在隐藏多线程设计中固有的许多复杂问题的同时提供多线程应用程序的优点。

[实现基于事件的异步模式](#)

描述将具有多种异步功能的类打包的标准方式。

[实现基于事件的异步模式的最佳做法](#)

描述根据基于事件的异步模式公开异步功能的要求。

[确定何时实现基于事件的异步模式](#)

描述如何确定何时应选择实现基于事件的异步模式而不是 [IAsyncResult](#) 模式。

[演练: 实现支持基于事件的异步模式的组件](#)

演示如何创建实现基于事件的异步模式的组件。它是使用 [System.ComponentModel](#) 命名空间的帮助器类实现的，这可确保该组件在任何应用程序模型下均可正常工作。

[如何: 使用支持基于事件的异步模式的组件](#)

描述如何使用支持基于事件的异步模式的组件。

参考

[AsyncOperation](#)

描述 **AsyncOperation** 类并提供指向其所有成员的链接。

[AsyncOperationManager](#)

描述 **AsyncOperationManager** 类并提供指向其所有成员的链接。

[BackgroundWorker](#)

描述 **BackgroundWorker** 组件并提供指向其所有成员的链接。

相关章节

[基于事件的异步模式技术示例](#)

演示如何使用基于事件的异步模式执行通用异步操作。

[多线程处理 \(Visual Basic\)](#)

描述 .NET Framework 中的多线程编程功能。

请参见

[概念](#)

[托管线程处理的最佳做法](#)

[事件和委托](#)

[其他资源](#)

[组件中的多线程处理](#)

[异步编程设计模式](#)

泛型(C# 编程指南)

泛型是 2.0 版 C# 语言和公共语言运行库 (CLR) 中的一个新功能。泛型将类型参数的概念引入 .NET Framework，类型参数使得设计如下类和方法成为可能：这些类和方法将一个或多个类型的指定推迟到客户端代码声明并实例化该类或方法的时候。例如，通过使用泛型类型参数 T，您可以编写其他客户端代码能够使用的单个类，而不致引入运行时强制转换或装箱操作的成本或风险，如下所示：

C#

```
// Declare the generic class
public class GenericList<T>
{
    void Add(T input) { }
}
class TestGenericList
{
    private class ExampleClass { }
    static void Main()
    {
        // Declare a list of type int
        GenericList<int> list1 = new GenericList<int>();

        // Declare a list of type string
        GenericList<string> list2 = new GenericList<string>();

        // Declare a list of type ExampleClass
        GenericList<ExampleClass> list3 = new GenericList<ExampleClass>();
    }
}
```

泛型概述

- 使用泛型类型可以最大限度地重用代码、保护类型的安全以及提高性能。
- 泛型最常见的用途是创建集合类。
- .NET Framework 类库在 [System.Collections.Generic](#) 命名空间中包含几个新的泛型集合类。应尽可能地使用这些类来代替普通的类，如 [System.Collections](#) 命名空间中的 [ArrayList](#)。
- 您可以创建自己的泛型接口、泛型类、泛型方法、泛型事件和泛型委托。
- 可以对泛型类进行约束以访问特定数据类型的方法。
- 关于泛型数据类型中使用的类型的信息可在运行时通过反射获取。

相关章节

更多信息：

- [泛型介绍\(C# 编程指南\)](#)
- [泛型的优点\(C# 编程指南\)](#)
- [泛型类型参数\(C# 编程指南\)](#)
- [类型参数的约束\(C# 编程指南\)](#)
- [泛型类\(C# 编程指南\)](#)
- [泛型接口\(C# 编程指南\)](#)
- [泛型方法\(C# 编程指南\)](#)
- [泛型委托\(C# 编程指南\)](#)
- [泛型代码中的默认关键字\(C# 编程指南\)](#)

- C++ 模板和 C# 泛型之间的区别(C# 编程指南)
- 泛型和反射(C# 编程指南)
- 运行库中的泛型(C# 编程指南)
- .NET Framework 类库中的泛型(C# 编程指南)
- “泛型”示例 (C#)

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 20 泛型

请参见

参考

[数据类型\(C# 编程指南\)](#)

[<typeparam>\(C# 编程指南\)](#)

[<typeparamref>\(C# 编程指南\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

泛型介绍 (C# 编程指南)

泛型类和泛型方法同时具备可重用性、类型安全和效率，这是非泛型类和非泛型方法无法具备的。泛型通常用在集合和在集合上运行的方法中。.NET Framework 2.0 版类库提供一个新的命名空间 [System.Collections.Generic](#)，其中包含几个新的基于泛型的集合类。建议面向 2.0 版的所有应用程序都使用新的泛型集合类，而不要使用旧的非泛型集合类，如 [ArrayList](#)。有关更多信息，请参见 [.NET Framework 类库中的泛型 \(C# 编程指南\)](#)。

当然，也可以创建自定义泛型类型和方法，以提供自己的通用解决方案，设计类型安全的高效模式。下面的代码示例演示一个用于演示用途的简单泛型链接列表类。(大多数情况下，建议使用 .NET Framework 类库提供的 [List<T>](#) 类，而不要自行创建类。) 在通常使用具体类型来指示列表中所存储项的类型时，可使用类型参数 *T*。其使用方法如下：

- 在 **AddHead** 方法中作为方法参数的类型。
- 在 **Node** 嵌套类中作为公共方法 **GetNext** 和 **Data** 属性的返回类型。
- 在嵌套类中作为私有成员数据的类型。

注意，*T* 可用于 **Node** 嵌套类。如果使用具体类型实例化 [GenericList<T>](#)(例如，作为 [GenericList<int>](#))，则所有的 *T* 都将被替换为 *int*。

C#

```
// type parameter T in angle brackets
public class GenericList<T>
{
    // The nested class is also generic on T
    private class Node
    {
        // T used in non-generic constructor
        public Node(T t)
        {
            next = null;
            data = t;
        }

        private Node next;
        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        // T as private member data type
        private T data;

        // T as return type of property
        public T Data
        {
            get { return data; }
            set { data = value; }
        }
    }

    private Node head;

    // constructor
    public GenericList()
    {
        head = null;
    }

    // T as method parameter type:
    public void AddHead(T t)
    {
        Node n = new Node(t);
        n.Next = head;
    }
}
```

```
        head = n;
    }

    public IEnumerator<T> GetEnumerator()
    {
        Node current = head;

        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }
}
```

下面的代码示例演示客户端代码如何使用泛型 `GenericList<T>` 类来创建整数列表。只需更改类型参数，即可方便地修改下面的代码示例，创建字符串或任何其他自定义类型的列表：

C#

```
class TestGenericList
{
    static void Main()
    {
        // int is the type argument
        GenericList<int> list = new GenericList<int>();

        for (int x = 0; x < 10; x++)
        {
            list.AddHead(x);
        }

        foreach (int i in list)
        {
            System.Console.Write(i + " ");
        }
        System.Console.WriteLine("\nDone");
    }
}
```

请参见

参考

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[泛型 \(C# 编程指南\)](#)

泛型的优点 (C# 编程指南)

在公共语言运行库和 C# 语言的早期版本中，通用化是通过在类型与通用基类型 `Object` 之间进行强制转换来实现的，泛型提供了针对这种限制的解决方案。通过创建泛型类，您可以创建一个在编译时类型安全的集合。

使用非泛型集合类的限制可以通过编写一小段程序来演示，该程序利用 .NET Framework 基类库中的 `ArrayList` 集合类。`ArrayList` 是一个使用起来非常方便的集合类，无需进行修改即可用来存储任何引用或值类型。

C#

```
// The .NET Framework 1.1 way to create a list:
System.Collections.ArrayList list1 = new System.Collections.ArrayList();
list1.Add(3);
list1.Add(105);

System.Collections.ArrayList list2 = new System.Collections.ArrayList();
list2.Add("It is raining in Redmond.");
list2.Add("It is snowing in the mountains.");
```

但这种方便是需要付出代价的。添加到 `ArrayList` 中的任何引用或值类型都将隐式地向上强制转换为 `Object`。如果项是值类型，则必须在将其添加到列表中时进行装箱操作，在检索时进行取消装箱操作。强制转换以及装箱和取消装箱操作都会降低性能；在必须对大型集合进行循环访问的情况下，装箱和取消装箱的影响非常明显。

另一个限制是缺少编译时类型检查；因为 `ArrayList` 将把所有项都强制转换为 `Object`，所以在编译时无法防止客户端代码执行以下操作：

C#

```
System.Collections.ArrayList list = new System.Collections.ArrayList();
// Add an integer to the list.
list.Add(3);
// Add a string to the list. This will compile, but may cause an error later.
list.Add("It is raining in Redmond.");

int t = 0;
// This causes an InvalidCastException to be returned.
foreach (int x in list)
{
    t += x;
}
```

尽管将字符串和 `ints` 组合在一个 `ArrayList` 中的做法在创建异类集合时是完全合法的，有时是有意图的，但这种做法更可能产生编程错误，并且直到运行时才能检测到此错误。

在 C# 语言的 1.0 和 1.1 版本中，只能通过编写自己的特定于类型的集合来避免 .NET Framework 基类库集合类中的通用代码的危险。当然，由于此类不可对多个数据类型重用，因此将丧失通用化的优点，并且您必须对要存储的每个类型重新编写该类。

`ArrayList` 和其他相似类真正需要的是：客户端代码基于每个实例指定这些类要使用的具体数据类型的方式。这样将不再需要向上强制转换为 `T:System.Object`，同时，也使得编译器可以进行类型检查。换句话说，`ArrayList` 需要一个 *type parameter*。这正是泛型所能提供的。在 `N:System.Collections.Generic` 命名空间的泛型 `List<T>` 集合中，向该集合添加项的操作类似于以下形式：

C#

```
// The .NET Framework 2.0 way to create a list
List<int> list1 = new List<int>();

// No boxing, no casting:
list1.Add(3);

// Compile-time error:
// list1.Add("It is raining in Redmond.");
```

对于客户端代码，与 **ArrayList** 相比，使用 `List<T>` 时添加的唯一语法是声明和实例化中的类型参数。虽然这稍微增加了些编码的复杂性，但好处是您可以创建一个比 **ArrayList** 更安全并且速度更快的列表，特别适用于列表项是值类型的情况。

[请参见](#)

[参考](#)

[泛型介绍\(C# 编程指南\)](#)

[System.Collections.Generic](#)

[概念](#)

[C# 编程指南](#)

泛型类型参数(C# 编程指南)

在泛型类型或方法定义中，类型参数是客户端在实例化泛型类型的变量时指定的特定类型的占位符。泛型类（如[泛型介绍\(C# 编程指南\)](#)中列出的 `GenericList<T>`）不可以像这样使用，因为它实际上并不是一个类型，而更像是一个类型的蓝图。若要使用 `GenericList<T>`，客户端代码必须通过指定尖括号中的类型参数来声明和实例化构造类型。此特定类的类型参数可以是编译器识别的任何类型。可以创建任意数目的构造类型实例，每个实例使用不同的类型参数，如下所示：

C#

```
GenericList<float> list1 = new GenericList<float>();
GenericList<ExampleClass> list2 = new GenericList<ExampleClass>();
GenericList<ExampleStruct> list3 = new GenericList<ExampleStruct>();
```

在每个 `GenericList<T>` 实例中，类中出现的每个 `T` 都会在运行时替换为相应的类型参数。通过这种替换方式，我们使用一个类定义创建了三个独立的类型安全的有效对象。有关 CLR 如何执行此替换的更多信息，请参见[运行库中的泛型\(C# 编程指南\)](#)。

类型参数命名准则

- 务必使用描述性名称命名泛型类型参数，除非单个字母名称完全可以让人了解它表示的含义，而描述性名称不会有更多的意义。

C#

```
public interface ISessionChannel<TSession> { /*...*/ }
public delegate TOutput Converter<TInput, TOutput>(TInput from);
public class List<T> { /*...*/ }
```

- 考虑使用 `T` 作为具有单个字母类型参数的类型的类型参数名。

C#

```
public int IComparer<T>() { return 0; }
public delegate bool Predicate<T>(T item);
public struct Nullable<T> where T : struct { /*...*/ }
```

- 务必将“`T`”作为描述性类型参数名的前缀。

C#

```
public interface ISessionChannel<TSession>
{
    TSession Session { get; }
}
```

- 考虑在参数名中指示对此类型参数的约束。例如，可以将带有 `ISession` 约束的参数命名为 `TSession`。

[请参见](#)

[参考](#)

[C++ 模板和 C# 泛型之间的区别\(C# 编程指南\)](#)

[System.Collections.Generic](#)

[概念](#)

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

类型参数的约束(C# 编程指南)

在定义泛型类时，可以对客户端代码能够在实例化类时用于类型参数的类型种类施加限制。如果客户端代码尝试使用某个约束所不允许的类型来实例化类，则会产生编译时错误。这些限制称为约束。约束是使用 **where** 上下文关键字指定的。下表列出了六种类型的约束：

约束	说明
T:结构	类型参数必须是值类型。可以指定除 Nullable 以外的任何值类型。有关更多信息，请参见 使用可空类型(C# 编程指南) 。
T:类	类型参数必须是引用类型，包括任何类、接口、委托或数组类型。
T:new()	类型参数必须具有无参数的公共构造函数。当与其他约束一起使用时， new() 约束必须最后指定。
T:<基类名>	类型参数必须是指定的基类或派生自指定的基类。
T:<接口名称>	类型参数必须是指定的接口或实现指定的接口。可以指定多个接口约束。约束接口也可以是泛型的。
T:U	为 T 提供的类型参数必须是为 U 提供的参数或派生自为 U 提供的参数。这称为裸类型约束。

使用约束的原因

如果要检查泛型列表中的某个项以确定它是否有效，或者将它与其他某个项进行比较，则编译器必须在一定程度上保证它需要调用的运算符或方法将受到客户端代码可能指定的任何类型参数的支持。这种保证是通过对泛型类定义应用一个或多个约束获得的。例如，基类约束告诉编译器：仅此类型的对象或从此类型派生的对象才可用作类型参数。一旦编译器有了这个保证，它就能够允许在泛型类中调用该类型的方法。约束是使用上下文关键字 **where** 应用的。下面的代码示例演示可通过应用基类约束添加到 `GenericList<T>` 类(在[泛型介绍\(C# 编程指南\)](#)中)的功能。

C#

```
public class Employee
{
    private string name;
    private int id;

    public Employee(string s, int i)
    {
        name = s;
        id = i;
    }

    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    public int ID
    {
        get { return id; }
        set { id = value; }
    }
}

public class GenericList<T> where T : Employee
{
    private class Node
    {
        private Node next;
        private T data;
    }
}
```

```

public Node(T t)
{
    next = null;
    data = t;
}

public Node Next
{
    get { return next; }
    set { next = value; }
}

public T Data
{
    get { return data; }
    set { data = value; }
}

private Node head;

public GenericList() //constructor
{
    head = null;
}

public void AddHead(T t)
{
    Node n = new Node(t);
    n.Next = head;
    head = n;
}

public IEnumerator<T> GetEnumerator()
{
    Node current = head;

    while (current != null)
    {
        yield return current.Data;
        current = current.Next;
    }
}

public T FindFirstOccurrence(string s)
{
    Node current = head;
    T t = null;

    while (current != null)
    {
        //The constraint enables access to the Name property.
        if (current.Data.Name == s)
        {
            t = current.Data;
            break;
        }
        else
        {
            current = current.Next;
        }
    }
    return t;
}
}

```

约束使得泛型类能够使用 Employee.Name 属性，因为类型为 T 的所有项都保证是 Employee 对象或从 Employee 继承的对象。

可以对同一类型参数应用多个约束，并且约束自身可以是泛型类型，如下所示：

C#

```
class EmployeeList<T> where T : Employee, IEmployee, System.IComparable<T>, new()
{
    // ...
}
```

通过约束类型参数，可以增加约束类型及其继承层次结构中的所有类型所支持的允许操作和方法调用的数量。因此，在设计泛型类或方法时，如果要对泛型成员执行除简单赋值之外的任何操作或调用 **System.Object** 不支持的任何方法，您将需要对该类型参数应用约束。

在应用 `where T : class` 约束时，建议不要对类型参数使用 `==` 和 `!=` 运算符，因为这些运算符仅测试引用同一性而不测试值相等性。即使在用作参数的类型中重载这些运算符也是如此。下面的代码说明了这一点：即使 `String` 类重载 `==` 运算符，输出也为 `false`。

C#

```
public static void OpTest<T>(T s, T t) where T : class
{
    System.Console.WriteLine(s == t);
}
static void Main()
{
    string s1 = "foo";
    System.Text.StringBuilder sb = new System.Text.StringBuilder("foo");
    string s2 = sb.ToString();
    OpTest<string>(s1, s2);
}
```

这种情况的原因在于，编译器在编译时仅知道 `T` 是引用类型，因此必须使用对所有引用类型都有效的默认运算符。如果需要测试值相等性，建议的方法是同时应用 `where T : IComparable<T>` 约束，并在将用于构造泛型类的任何类中实现该接口。

未绑定的类型参数

没有约束的类型参数（如公共类 `SampleClass<T>{}` 中的 `T`）称为未绑定的类型参数。未绑定的类型参数具有以下规则：

- 不能使用 `!=` 和 `==` 运算符，因为无法保证具体类型参数能支持这些运算符。
- 可以在它们与 **System.Object** 之间来回转换，或将它们显式转换为任何接口类型。
- 可以将它们与 `null` 进行比较。将未绑定的参数与 `null` 进行比较时，如果类型参数为值类型，则该比较将始终返回 `false`。

裸类型约束

用作约束的泛型类型参数称为裸类型约束。当具有自己的类型参数的成员函数需要将该参数约束为包含类型的类型参数时，裸类型约束很有用，如下面的示例所示：

C#

```
class List<T>
{
    void Add<U>(List<U> items) where U : T /*...*/
}
```

在上面的示例中，`T` 在 `Add` 方法的上下文中是一个裸类型约束，而在 `List` 类的上下文中是一个未绑定的类型参数。

裸类型约束还可以在泛型类定义中使用。注意，还必须已经和其他任何类型参数一起在尖括号中声明了裸类型约束：

C#

```
//naked type constraint
public class SampleClass<T, U, V> where T : V { }
```

泛型类的裸类型约束的作用非常有限，因为编译器除了假设某个裸类型约束派生自 **System.Object** 以外，不会做其他任何假设。在希望强制两个类型参数之间的继承关系的情况下，可对泛型类使用裸类型约束。

请参见

参考

[泛型介绍 \(C# 编程指南\)](#)

[new 约束 \(C# 参考\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

泛型类(C# 编程指南)

泛型类封装不是特定于具体数据类型的操作。泛型类最常用于集合，如链接列表、哈希表、堆栈、队列、树等，其中，像从集合中添加和移除项这样的操作都以大体上相同的方式执行，与所存储数据的类型无关。

对于大多数需要集合类的方案，推荐的方法是使用 .NET Framework 2.0 类库中所提供的类。有关使用这些类的更多信息，请参见 [.NET Framework 类库中的泛型\(C# 编程指南\)](#)。

一般情况下，创建泛型类的过程为：从一个现有的具体类开始，逐一将每个类型更改为类型参数，直至达到通用化和可用性的最佳平衡。创建您自己的泛型类时，需要特别注意以下事项：

- 将哪些类型通用化为类型参数。

一般规则是，能够参数化的类型越多，代码就会变得越灵活，重用性就越好。但是，太多的通用化会使其他开发人员难以阅读或理解代码。

- 如果存在约束，应对类型参数应用什么约束(请参见[类型参数的约束\(C# 编程指南\)](#))。

一个有用的规则是，应用尽可能最多的约束，但仍使您能够处理需要处理的类型。例如，如果您知道您的泛型类仅用于引用类型，则应用类约束。这可以防止您的类被意外地用于值类型，并允许您对 `T` 使用 `as` 运算符以及检查空值。

- 是否将泛型行为分解为基类和子类。

由于泛型类可以作为基类使用，此处适用的设计注意事项与非泛型类相同。有关从泛型基类继承的规则，请参见下面的内容。

- 是否实现一个或多个泛型接口。

例如，如果您设计一个类，该类将用于创建基于泛型的集合中的项，则可能需要实现一个接口，如 `IComparable<T>`，其中 `T` 是您的类的类型。

有关简单泛型类的示例，请参见[泛型介绍\(C# 编程指南\)](#)

类型参数和约束的规则对于泛型类行为有几方面的含义，特别是关于继承和成员可访问性。请务必先理解一些术语，然后再继续进行。对于泛型类 `Node<T>`，客户端代码可以通过指定类型参数引用该类，以创建封闭式构造类型 (`Node<int>`)，或者可以让类型参数处于未指定状态(例如在指定泛型基类时)以创建开放式构造类型 (`Node<T>`)。泛型类可以从具体的、封闭式构造或开放式构造基类继承：

C#

```
class BaseNode { }
class BaseNodeGeneric<T> { }

// concrete type
class NodeConcrete<T> : BaseNode { }

//closed constructed type
class NodeClosed<T> : BaseNodeGeneric<int> { }

//open constructed type
class NodeOpen<T> : BaseNodeGeneric<T> { }
```

非泛型(具体)类可以从封闭式构造基类继承，但无法从开放式构造类或裸类型参数继承，因为在运行时客户端代码无法提供实例化基类所需的类型变量。

C#

```
//No error
class Node1 : BaseNodeGeneric<int> { }

//Generates an error
//class Node2 : BaseNodeGeneric<T> {}

//Generates an error
//class Node3 : T {}
```

从开放式构造类型继承的泛型类必须为任何未被继承类共享的基类型参数提供类型变量，如以下代码所示：

C#

```
class BaseNodeMultiple<T, U> { }

//No error
class Node4<T> : BaseNodeMultiple<T, int> { }

//No error
class Node5<T, U> : BaseNodeMultiple<T, U> { }

//Generates an error
//class Node6<T> : BaseNodeMultiple<T, U> {}
```

从开放式构造类型继承的泛型类必须指定约束，这些约束是基类型约束的超集或暗示基类型约束：

C#

```
class NodeItem<T> where T : System.IComparable<T>, new() { }
class SpecialNodeItem<T> : NodeItem<T> where T : System.IComparable<T>, new() { }
```

泛型类型可以使用多个类型参数和约束，如下所示：

C#

```
class SuperKeyType<K, V, U>
    where U : System.IComparable<U>
    where V : new()
{ }
```

开放式构造类型和封闭式构造类型可以用作方法参数：

C#

```
void Swap<T>(List<T> list1, List<T> list2)
{
    //code to swap items
}

void Swap(List<int> list1, List<int> list2)
{
    //code to swap items
}
```

泛型类是不变的。也就是说，如果输入参数指定 `List<BaseClass>`，则当您试图提供 `List<DerivedClass>` 时，将发生编译时错误。

请参见

参考

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

泛型接口 (C# 编程指南)

为泛型集合类或表示集合中项的泛型类定义接口通常很有用。对于泛型类，使用泛型接口十分可取，例如使用 `IComparable<T>` 而不使用 `IComparable`，这样可以避免值类型的装箱和取消装箱操作。<.NET Framework 2.0 类库定义了若干新的泛型接口，以用于 `System.Collections.Generic` 命名空间中新的集合类。

将接口指定为类型参数的约束时，只能使用实现此接口的类型。下面的代码示例显示从 `GenericList<T>` 类派生的 `SortedList<T>` 类。有关更多信息，请参见 [泛型介绍 \(C# 编程指南\)](#)。`SortedList<T>` 添加了约束 `where T : IComparable<T>`。这将使 `SortedList<T>` 中的 **BubbleSort** 方法能够对列表元素使用泛型 `CompareTo` 方法。在此示例中，列表元素为简单类，即实现 `IComparable<Person>` 的 `Person`。

C#

```
//Type parameter T in angle brackets.
public class GenericList<T> : System.Collections.Generic.IEnumerable<T>
{
    protected Node head;
    protected Node current = null;

    // Nested class is also generic on T
    protected class Node
    {
        public Node next;
        private T data; //T as private member datatype

        public Node(T t) //T used in non-generic constructor
        {
            next = null;
            data = t;
        }

        public Node Next
        {
            get { return next; }
            set { next = value; }
        }

        public T Data //T as return type of property
        {
            get { return data; }
            set { data = value; }
        }
    }

    public GenericList() //constructor
    {
        head = null;
    }

    public void AddHead(T t) //T as method parameter type
    {
        Node n = new Node(t);
        n.Next = head;
        head = n;
    }

    // Implementation of the iterator
    public System.Collections.Generic.IEnumerator<T> GetEnumerator()
    {
        Node current = head;
        while (current != null)
        {
            yield return current.Data;
            current = current.Next;
        }
    }
}
```

```

// IEnumerable<T> inherits from IEnumerable, therefore this class
// must implement both the generic and non-generic versions of
// GetEnumerator. In most cases, the non-generic method can
// simply call the generic method.
System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator()
{
    return GetEnumerator();
}
}

public class SortedList<T> : GenericList<T> where T : System.IComparable<T>
{
    // A simple, unoptimized sort algorithm that
    // orders list elements from lowest to highest:

    public void BubbleSort()
    {
        if (null == head || null == head.Next)
        {
            return;
        }
        bool swapped;

        do
        {
            Node previous = null;
            Node current = head;
            swapped = false;

            while (current.next != null)
            {
                // Because we need to call this method, the SortedList
                // class is constrained on IEnumerable<T>
                if (current.Data.CompareTo(current.next.Data) > 0)
                {
                    Node tmp = current.next;
                    current.next = current.next.next;
                    tmp.next = current;

                    if (previous == null)
                    {
                        head = tmp;
                    }
                    else
                    {
                        previous.next = tmp;
                    }
                    previous = tmp;
                    swapped = true;
                }
                else
                {
                    previous = current;
                    current = current.next;
                }
            }
        } while (swapped);
    }
}

// A simple class that implements IComparable<T> using itself as the
// type argument. This is a common design pattern in objects that
// are stored in generic lists.
public class Person : System.IComparable<Person>
{
    string name;
    int age;
}

```

```

public Person(string s, int i)
{
    name = s;
    age = i;
}

// This will cause list elements to be sorted on age values.
public int CompareTo(Person p)
{
    return age - p.age;
}

public override string ToString()
{
    return name + ":" + age;
}

// Must implement Equals.
public bool Equals(Person p)
{
    return (this.age == p.age);
}

class Program
{
    static void Main()
    {
        //Declare and instantiate a new generic SortedList class.
        //Person is the type argument.
        SortedList<Person> list = new SortedList<Person>();

        //Create name and age values to initialize Person objects.
        string[] names = new string[]
        {
            "Franscoise",
            "Bill",
            "Li",
            "Sandra",
            "Gunnar",
            "Alok",
            "Hiroyuki",
            "Maria",
            "Alessandro",
            "Raul"
        };

        int[] ages = new int[] { 45, 19, 28, 23, 18, 9, 108, 72, 30, 35 };

        //Populate the list.
        for (int x = 0; x < 10; x++)
        {
            list.AddHead(new Person(names[x], ages[x]));
        }

        //Print out unsorted list.
        foreach (Person p in list)
        {
            System.Console.WriteLine(p.ToString());
        }
        System.Console.WriteLine("Done with unsorted list");

        //Sort the list.
        list.BubbleSort();

        //Print out sorted list.
        foreach (Person p in list)
    }
}

```

```

    {
        System.Console.WriteLine(p.ToString());
    }
    System.Console.WriteLine("Done with sorted list");
}
}

```

可将多重接口指定为单个类型上的约束，如下所示：

C#

```

class Stack<T> where T : System.IComparable<T>, IEnumerable<T>
{
}

```

一个接口可定义多个类型参数，如下所示：

C#

```

interface IDictionary<K, V>
{
}

```

类之间的继承规则同样适用于接口：

C#

```

interface IMonth<T> { }

interface IJanuary : IMonth<int> { } //No error
interface IFebruary<T> : IMonth<int> { } //No error
interface IMarch<T> : IMonth<T> { } //No error
//interface IApril<T> : IMonth<T, U> {} //Error

```

如果泛型接口为逆变的，即仅使用其类型参数作为返回值，则此泛型接口可以从非泛型接口继承。在 .NET Framework 类库中，`IEnumerable<T>` 从 `IEnumerable` 继承，因为 `IEnumerable<T>` 仅在 `GetEnumerator` 的返回值和当前属性 getter 中使用 `T`。

具体类可以实现已关闭的构造接口，如下所示：

C#

```

interface IBaseInterface<T> { }

class SampleClass : IBaseInterface<string> { }

```

只要类参数列表提供了接口必需的所有参数，泛型类便可以实现泛型接口或已关闭的构造接口，如下所示：

C#

```

interface IBaseInterface1<T> { }
interface IBaseInterface2<T, U> { }

class SampleClass1<T> : IBaseInterface1<T> { } //No error
class SampleClass2<T> : IBaseInterface2<T, string> { } //No error

```

对于泛型类、泛型结构或泛型接口中的方法，控制方法重载的规则相同。有关更多信息，请参见[泛型方法\(C# 编程指南\)](#)。

请参见

参考

[泛型介绍\(C# 编程指南\)](#)

[接口](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[.NET Framework 中的泛型](#)

泛型方法 (C# 编程指南)

泛型方法是使用类型参数声明的方法，如下所示：

C#

```
static void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

下面的代码示例演示一种使用 int 作为类型参数的方法调用方式：

C#

```
public static void TestSwap()
{
    int a = 1;
    int b = 2;

    Swap<int>(ref a, ref b);
    System.Console.WriteLine(a + " " + b);
}
```

也可以省略类型参数，编译器将推断出该参数。下面对 Swap 的调用等效于前面的调用：

C#

```
Swap(ref a, ref b);
```

相同的类型推断规则也适用于静态方法以及实例方法。编译器能够根据传入的方法参数推断类型参数；它无法仅从约束或返回值推断类型参数。因此，类型推断不适用于没有参数的方法。类型推断在编译时、编译器尝试解析任何重载方法签名之前进行。编译器向共享相同名称的所有泛型方法应用类型推断逻辑。在重载解析步骤中，编译器仅包括类型推断取得成功的那些泛型方法。

在泛型类中，非泛型方法可以访问类级别类型参数，如下所示：

C#

```
class SampleClass<T>
{
    void Swap(ref T lhs, ref T rhs) { }
}
```

如果定义的泛型方法接受与包含类相同的类型参数，编译器将生成警告 CS0693，因为在方法范围内，为内部 T 提供的参数将隐藏为外部 T 提供的参数。除了类初始化时提供的类型参数之外，如果需要灵活调用具有类型参数的泛型类方法，请考虑为方法的类型参数提供其他标识符，如下面示例中的 GenericList2<T> 所示。

C#

```
class GenericList<T>
{
    // CS0693
    void SampleMethod<T>() { }

    class GenericList2<T>
    {
```

```
//No warning
void SampleMethod<U>() { }
```

使用约束对方法中的类型参数启用更专门的操作。此版本的 `Swap<T>` 现在称为 `SwapIfGreater<T>`，它只能与实现 `IComparable<T>` 的类型参数一起使用。

C#

```
void SwapIfGreater<T>(ref T lhs, ref T rhs) where T : System.IComparable<T>
{
    T temp;
    if (lhs.CompareTo(rhs) > 0)
    {
        temp = lhs;
        lhs = rhs;
        rhs = temp;
    }
}
```

泛型方法可以使用许多类型参数进行重载。例如，下列方法可以全部存在于同一个类中：

C#

```
void DoWork() { }
void DoWork<T>() { }
void DoWork<T, U>() { }
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 20.6.4 类型参数的推理。

请参见

参考

[泛型介绍 \(C# 编程指南\)](#)

[方法 \(C# 编程指南\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

泛型和数组 (C# 编程指南)

在 C# 2.0 中，下限为零的一维数组自动实现 `IList<T>`。这使您可以创建能够使用相同代码循环访问数组和其他集合类型的泛型方法。此技术主要对读取集合中的数据很有用。`IList<T>` 接口不能用于在数组中添加或移除元素；如果试图在此上下文中调用 `IList<T>` 方法（如数组的 `RemoveAt`），将引发异常。

下面的代码示例演示带有 `IList<T>` 输入参数的单个泛型方法如何同时循环访问列表和数组，本例中为整数数组。

C#

```
class Program
{
    static void Main()
    {
        int[] arr = { 0, 1, 2, 3, 4 };
        List<int> list = new List<int>();

        for (int x = 5; x < 10; x++)
        {
            list.Add(x);
        }

        ProcessItems<int>(arr);
        ProcessItems<int>(list);
    }

    static void ProcessItems<T>(IList<T> coll)
    {
        foreach (T item in coll)
        {
            System.Console.Write(item.ToString() + " ");
        }
        System.Console.WriteLine();
    }
}
```

注意

尽管 `ProcessItems` 方法无法添加或移除项，但对于 `ProcessItems` 内部的 `T[]`，`IsReadOnly` 属性返回 `False`，因为该数组本身未声明 `ReadOnly` 特性。

请参见

参考

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[泛型 \(C# 编程指南\)](#)

[数组 \(C# 编程指南\)](#)

其他资源

[.NET Framework 中的泛型](#)

泛型委托(C# 编程指南)

委托可以定义自己的类型参数。引用泛型委托的代码可以指定类型参数以创建已关闭的构造类型，就像实例化泛型类或调用泛型方法一样，如下例所示：

C#

```
public delegate void Del<T>(T item);
public static void Notify(int i) { }

Del<int> m1 = new Del<int>(Notify);
```

C# 2.0 版具有称为方法组转换的新功能，此功能适用于具体委托类型和泛型委托类型，并使您可以使用如下简化的语法写入上一行：

C#

```
Del<int> m2 = Notify;
```

在泛型类内部定义的委托使用泛型类类型参数的方式可以与类方法所使用的方式相同。

C#

```
class Stack<T>
{
    T[] items;
    int index;

    public delegate void StackDelegate(T[] items);
}
```

引用委托的代码必须指定包含类的类型变量，如下所示：

C#

```
private static void DoWork(float[] items) { }

public static void TestStack()
{
    Stack<float> s = new Stack<float>();
    Stack<float>.StackDelegate d = DoWork;
}
```

根据典型设计模式定义事件时，泛型委托尤其有用，因为发送方参数可以为强类型，不再需要强制转换成 [Object](#)，或反向强制转换。

C#

```
delegate void StackEventHandler<T, U>(T sender, U eventArgs);

class Stack<T>
{
    public class StackEventArgs : System.EventArgs { }
    public event StackEventHandler<Stack<T>, StackEventArgs> stackEvent;

    protected virtual void OnStackChanged(StackEventArgs a)
    {
        stackEvent(this, a);
    }
}
```

```
class SampleClass
{
    public void HandleStackChange<T>(Stack<T> stack, Stack<T>.StackEventArgs args) { }

    public static void Test()
    {
        Stack<double> s = new Stack<double>();
        SampleClass o = new SampleClass();
        s.stackEvent += o.HandleStackChange;
    }
}
```

请参见

参考

[泛型介绍 \(C# 编程指南\)](#)

[泛型方法 \(C# 编程指南\)](#)

[泛型类 \(C# 编程指南\)](#)

[泛型接口 \(C# 编程指南\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[委托 \(C# 编程指南\)](#)

其他资源

[.NET Framework 中的泛型](#)

泛型代码中的默认关键字(C# 编程指南)

在泛型类和泛型方法中产生的一个问题是在预先未知以下情况时，如何将默认值分配给参数化类型 T：

- T 是引用类型还是值类型。
- 如果 T 为值类型，则它是数值还是结构。

给定参数化类型 T 的一个变量 t，只有当 T 为引用类型时，语句 `t = null` 才有效；只有当 T 为数值类型而不是结构时，语句 `t = 0` 才能正常使用。解决方案是使用 **default** 关键字，此关键字对于引用类型会返回空，对于数值类型会返回零。对于结构，此关键字将返回初始化为零或空的每个结构成员，具体取决于这些结构是值类型还是引用类型。以下来自 `GenericList<T>` 类的示例显示了如何使用 **default** 关键字。有关更多信息，请参见[泛型概述](#)。

C#

```
public class GenericList<T>
{
    private class Node
    {
        //...

        public Node Next;
        public T Data;
    }

    private Node head;

    //...

    public T GetNext()
    {
        T temp = default(T);

        Node current = head;
        if (current != null)
        {
            temp = current.Data;
            current = current.Next;
        }
        return temp;
    }
}
```

[请参见](#)

[参考](#)

[泛型方法\(C# 编程指南\)](#)

[System.Collections.Generic](#)

[概念](#)

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

[其他资源](#)

[.NET Framework 中的泛型](#)

C++ 模板和 C# 泛型之间的区别(C# 编程指南)

C# 泛型和 C++ 模板都是用于提供参数化类型支持的语言功能。然而，这两者之间存在许多差异。在语法层面上，C# 泛型是实现参数化类型的更简单方法，不具有 C++ 模板的复杂性。此外，C# 并不尝试提供 C++ 模板所提供的所有功能。在实现层面，主要区别在于，C# 泛型类型替换是在运行时执行的，从而为实例化的对象保留了泛型类型信息。有关更多信息，请参见[运行库中的泛型\(C# 编程指南\)](#)。

以下是 C# 泛型和 C++ 模板之间的主要差异：

- C# 泛型未提供与 C++ 模板相同程度的灵活性。例如，尽管在 C# 泛型类中可以调用用户定义的运算符，但不能调用算术运算符。
- C# 不允许非类型模板参数，如 `template C<int i> {}`。
- C# 不支持显式专用化，即特定类型的模板的自定义实现。
- C# 不支持部分专用化：类型参数子集的自定义实现。
- C# 不允许将类型参数用作泛型类型的基类。
- C# 不允许类型参数具有默认类型。
- 在 C# 中，尽管构造类型可用作泛型，但泛型类型参数自身不能是泛型。C++ 确实允许模板参数。
- C++ 允许那些可能并非对模板中的所有类型参数都有效的代码，然后将检查该代码中是否有用作类型参数的特定类型。C# 要求相应地编写类中的代码，使之能够使用任何满足约束的类型。例如，可以在 C++ 中编写对类型参数的对象使用算术运算符 + 和 - 的函数，这会在使用不支持这些运算符的类型来实例化模板时产生错误。C# 不允许这样；唯一允许的语言构造是那些可从约束推导出来的构造。

请参见

参考

[泛型介绍\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Templates](#)

运行库中的泛型(C# 编程指南)

将泛型类型或方法编译为 Microsoft 中间语言 (MSIL) 时，它包含将其标识为具有类型参数的元数据。泛型类型的 MSIL 的使用因所提供的类型参数是值类型还是引用类型而不同。

第一次用值类型作为参数来构造泛型类型时，运行库会创建专用泛型类型，将提供的参数代入到 MSIL 中的适当位置。对于每个用作参数的唯一值类型，都会创建一次专用泛型类型。

例如，假设您的程序代码声明了一个由整数构造的堆栈，如下所示：

C#

```
Stack<int> stack;
```

在此位置，运行库生成 `Stack <T>` 类的专用版本，并相应地用整数替换其参数。现在，只要程序代码使用整数堆栈，运行库就会重用生成的专用 `Stack<T>` 类。在下面的示例中，创建了整数堆栈的两个实例，它们共享 `Stack<int>` 代码的单个实例：

C#

```
Stack<int> stackOne = new Stack<int>();
Stack<int> stackTwo = new Stack<int>();
```

但是，如果在程序代码中的其他位置创建了另一个 `Stack<T>` 类，这次使用不同的值类型（如 `long` 或用户定义的结构）作为其参数，则运行库会生成泛型类型的另一版本（这次将在 MSIL 中的适当位置代入 `long`）。由于每个专用泛型类本身就包含值类型，因此不再需要转换。

对于引用类型，泛型的工作方式略有不同。第一次使用任何引用类型构造泛型类型时，运行库会创建专用泛型类型，用对象引用替换 MSIL 中的参数。然后，每次使用引用类型作为参数来实例化构造类型时，无论引用类型的具体类型是什么，运行库都会重用以前创建的泛型类型的专用版本。之所以可以这样，是因为所有引用的大小相同。

例如，假设您有两个引用类型：一个 `Customer` 类和一个 `Order` 类，并且进一步假设您创建了一个 `Customer` 类型的堆栈：

C#

```
class Customer { }
class Order { }
```

C#

```
Stack<Customer> customers;
```

在此情况下，运行库生成 `Stack<T>` 类的一个专用版本，该版本不是存储数据，而是存储稍后将填写的对象引用。假设下一行代码创建另一个引用类型的堆栈，称为 `Order`：

C#

```
Stack<Order> orders = new Stack<Order>();
```

不同于值类型，对于 `Order` 类型不创建 `Stack<T>` 类的另一个专用版本。而是创建 `Stack<T>` 类的一个专用版本实例，并将 `orders` 变量设置为引用它。假设接下来您遇到一行创建 `Customer` 类型堆栈的代码：

C#

```
customers = new Stack<Customer>();
```

与前面使用 `Order` 类型创建的 `Stack<T>` 类一样，创建了专用 `Stack<T>` 类的另一个实例，并且其中所包含的指针被设置为引用 `Customer` 类型大小的内存区域。因为引用类型的数量会随程序的不同而大幅变化，C# 泛型实现将编译器为引用类型的泛型类创建的专用类的数量减小到一个，从而大幅减小代码量的增加。

此外，使用类型参数实例化泛型 C# 类时，无论它是值类型还是引用类型，可以在运行时使用反射查询它，并且可以确定它的实际类型和类型参数。

请参见

参考

[泛型介绍 \(C# 编程指南\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

其他资源

[.NET Framework 中的泛型](#)

.NET Framework 类库中的泛型 (C# 编程指南)

.NET Framework 2.0 版类库提供一个新的命名空间 [System.Collections.Generic](#), 其中包括几个随时可用的泛型集合类和关联接口。其他命名空间(如 [System](#))也提供新的泛型接口, 如 [IComparable<T>](#)。与早期版本的 .NET Framework 所提供的非泛型集合类相比, 这些类和接口更为高效和类型安全。在设计和实现自己的自定义集合类之前, 请考虑是否能够使用基类库所提供的类, 或是否能从基类库所提供的类派生。

请参见

参考

[泛型介绍 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[何时使用泛型集合](#)

其他资源

[集合和数据结构](#)

泛型和反射 (C# 编程指南)

因为公共语言运行库 (CLR) 能够在运行时访问泛型类型信息，所以可以使用反射获取关于泛型类型的信息，方法与用于非泛型类型的方法相同。有关更多信息，请参见[运行库中的泛型 \(C# 编程指南\)](#)。

在 .NET Framework 2.0 中，[Type](#) 类增添了一个新成员以启用泛型类型的运行时信息。有关如何使用这些方法和属性的更多信息，请参见关于这些类的文档。[System.Reflection.Emit](#) 命名空间还包含支持泛型的新成员。请参见[如何：定义具有反射发出的泛型类型](#)。

有关泛型反射中使用的术语的固定条件列表，请参见 [IsGenericType](#) 属性备注。

System.Type 成员名称	说明
IsGenericType	如果类型为泛型，则返回 true。
GetGenericArguments	返回 Type 对象数组，这些对象表示为构造类型提供的类型变量，或泛型类型定义的类型参数。
GetGenericTypeDefinition	返回当前构造类型的基础泛型类型定义。
GetGenericParameterConstraints	返回表示当前泛型类型参数约束的 Type 对象的数组。
ContainsGenericParameters	如果类型或其任意封闭类型或方法包含没有被提供特定类型的类型参数，则返回 true。
GenericParameterAttributes	获取 GenericParameterAttributes 标志的组合，这些标志描述当前泛型类型参数的特殊约束。
GenericParameterPosition	对于表示类型参数的 Type 对象，获取类型参数在声明该类型参数的泛型类型定义或泛型方法定义的类型参数列表中的位置。
IsGenericParameter	获取一个值，该值指示当前 Type 是表示泛型类型定义的类型参数，还是泛型方法定义的类型参数。
IsGenericTypeDefinition	获取一个值，该值指示当前 Type 是否表示可以用来构造其他泛型类型的泛型类型定义。如果类型表示泛型类型的定义，则返回 true。
DeclaringMethod	返回定义当前泛型类型参数的泛型方法；如果类型参数不是由泛型方法定义的，则返回空值。
MakeGenericType	用类型数组的元素替代当前泛型类型定义的类型参数，并返回表示结果构造类型的 Type 对象。

此外，[MethodInfo](#) 类中还添加了新成员以启用泛型方法的运行时信息。有关泛型方法反射中使用的术语的固定条件列表，请参见 [IsGenericMethod](#) 属性备注。

System.Reflection.MemberInfo 成员名称	说明
IsGenericMethod	如果方法为泛型，则返回 true。
GetGenericArguments	返回 Type 对象数组，这些对象表示构造泛型方法的类型变量，或泛型方法定义的类型参数。
GetGenericMethodDefinition	返回当前构造方法的基础泛型方法定义。
ContainsGenericParameters	如果方法或其任意封闭类型包含没有被提供特定类型的任何类型参数，则返回 true。
IsGenericMethodDefinition	如果当前 MethodInfo 表示泛型方法的定义，则返回 true。

[MakeGenericMethod](#)

用类型数组的元素替代当前泛型方法定义的类型参数，并返回表示结果构造方法的 **MethodInfo** 对象。

请参见

概念

[C# 编程指南](#)

[泛型 \(C# 编程指南\)](#)

[反射和泛型概述](#)

其他资源

[.NET Framework 中的泛型](#)

泛型和属性(C# 编程指南)

属性可以应用于泛型类型中，方式与应用于非泛型类型相同。有关应用属性的更多信息，请参见[属性\(C# 编程指南\)](#)。

自定义属性只允许引用开放泛型类型(未提供类型参数的泛型类型)和封闭构造泛型类型(为所有类型参数提供参数)。

下面的示例使用此自定义属性：

C#

```
class CustomAttribute : System.Attribute
{
    public System.Object info;
}
```

属性可以引用开放式泛型类型：

C#

```
public class GenericClass1<T> { }

[CustomAttribute(info = typeof(GenericClass1<>))]
class ClassA { }
```

使用数目适当的若干个逗号指定多个类型参数。在此示例中，`GenericClass2` 有两个类型参数：

C#

```
public class GenericClass2<T, U> { }

[CustomAttribute(info = typeof(GenericClass2<, >))]
class ClassB { }
```

属性可以引用封闭式构造泛型类型：

C#

```
public class GenericClass3<T, U, V> { }

[CustomAttribute(info = typeof(GenericClass3<int, double, string>))]
class ClassC { }
```

引用泛型类型参数的属性将导致编译时错误：

C#

```
// [CustomAttribute(info = typeof(GenericClass3<int, T, string>))] //Error
class ClassD<T> { }
```

不能从 `Attribute` 继承泛型类型：

C#

```
// public class CustomAtt<T> : System.Attribute {} //Error
```

若要在运行时获得有关泛型类型或类型参数的信息，可以使用 `System.Reflection` 的方法。有关更多信息，请参见[泛型和反射\(C# 编程指南\)](#)

请参见

[概念](#)

[C# 编程指南](#)

[泛型 \(C# 编程指南\)](#)

[属性 \(Attribute\) 概述](#)

泛型类型中的变化(C# 编程指南)

在 C# 中添加泛型的一个主要好处是能够使用 `System.Collections.Generic` 命名空间中的类型，轻松地创建强类型集合。例如，您可以创建一个类型为 `List<int>` 的变量，编译器将检查对该变量的所有访问，确保只有 `ints` 添加到该集合中。与 C# 1.0 版中的非类型化集合相比，这是可用性方面的一个很大改进。

遗憾的是强类型集合有自身的缺陷。例如，假设您有一个强类型 `List<object>`，您想将 `List<int>` 中的所有元素追加到 `List<object>` 中。您可能希望编写如下代码：

C#

```
List<int> ints = new List<int>();
ints.Add(1);
ints.Add(10);
ints.Add(42);
List<object> objects = new List<object>();

// doesnt compile ints is not a IEnumerable<object>
// objects.AddRange(ints);
```

在这种情况下，您希望能够将 `List<int>`（它同时也是 `IEnumerable<int>`）当作 `IEnumerable<object>`。这样做看起来似乎很合理，因为 `int` 可以转换为对象。这与能够将 `string[]` 当作 `object[]`（现在你就可以这样做）非常相似。如果您正面临这种情况，那么您需要一种称为泛型变化的功能，它将泛型类型的一种实例化（在本例中为 `IEnumerable<int>`）当成该类型的另一种实例化（在本例中为 `IEnumerable<object>`）。

C# 不支持泛型类型的变化，所以当遇到这种情况时，您需要尝试可以用来解决此问题的几种技术，找到一种适合的技术。对于最简单的情况，例如上例中的单个方法 `AddRange`，您可以声明一个简单的帮助器方法来为您执行转换。例如，您可以编写如下方法：

C#

```
// Simple workaround for single method
// Variance in one direction only
public static void Add<S, D>(List<S> source, List<D> destination)
    where S : D
{
    foreach (S sourceElement in source)
    {
        destination.Add(sourceElement);
    }
}
```

它使您能够完成以下操作：

C#

```
// does compile
VarianceWorkaround.Add<int, object>(ints, objects);
```

此示例演示了一种简单的变化解决方法的一些特征。帮助器方法带两个类型参数，分别对应于源和目标，源类型参数 `S` 有一个约束，即目标类型参数 `D`。这意味着读取的 `List<S>` 所包含的元素必须可以转换为插入的 `List<D>` 类型的元素。这使编译器可以强制 `int` 必须可转换为对象。将类型参数约束为从另一类型参数派生被称为裸类型参数约束。

定义一个方法来解决变化问题不算是一种过于拙劣的方法。遗憾的是变化问题很快就会变得非常复杂。下一级别的复杂性产生在当您想要将一个实例化的接口当作另一个实例化的接口时。例如，您有一个 `IEnumerable<int>`，您想将它传递给一个只以 `IEnumerable<object>` 为参数的方法。同样，这样做也是有一定意义的，因为您可以将 `IEnumerable<object>` 看作对象的序列，将 `IEnumerable<int>` 看作 `ints` 的序列。由于 `ints` 是对象，因此 `ints` 的序列应当可以被当作对象序列。例如：

C#

```
static void PrintObjects(IEnumerable<object> objects)
{
```

```

foreach (object o in objects)
{
    Console.WriteLine(o);
}

```

您要调用的方法如下所示：

C#

```

// would like to do this, but can't ...
// ... ints is not an IEnumerable<object>
//PrintObjects(ints);

```

接口 case 的解决方法是：创建为接口的每个成员执行转换的包装对象。它看起来应如下所示：

C#

```

// Workaround for interface
// Variance in one direction only so type expressions are natural
public static IEnumerable<D> Convert<S, D>(IEnumerable<S> source)
    where S : D
{
    return new EnumerableWrapper<S, D>(source);
}

private class EnumerableWrapper<S, D> : IEnumerable<D>
    where S : D
{

```

它使您能够完成以下操作：

C#

```
PrintObjects(VarianceWorkaround.Convert<int, object>(ints));
```

同样，请注意包装类和帮助器方法的裸类型参数约束。此系统已经变得相当复杂，但是包装类中的代码非常简单；它只委托给所包装接口的成员，除了简单的类型转换外，不执行其他任何操作。为什么不让编译器允许从 `IEnumerable<int>` 直接转换为 `IEnumerable<object>` 呢？

尽管在查看集合的只读视图的情况下，变化是类型安全的，然而在同时涉及读写操作的情况下，变化不是类型安全的。例如，不能用此自动方法处理 `IList<T>` 接口。您仍然可以编写一个帮助器，用类型安全的方式包装 `IList<T>` 上的所有读操作，但是写操作的包装就不能如此简单了。

下面是处理 `IList<T>` 接口的变化的包装的一部分，它显示在读和写两个方向上的变化所引起的问题。

C#

```

private class ListWrapper<S, D> : CollectionWrapper<S, D>, IList<D>
    where S : D
{
    public ListWrapper(IList<S> source) : base(source)
    {
        this.source = source;
    }

    public int IndexOf(D item)
    {
        if (item is S)
        {
            return this.source.IndexOf((S) item);
        }
        else
        {

```

```

        return -1;
    }

    // variance the wrong way ...
    // ... can throw exceptions at runtime
    public void Insert(int index, D item)
    {
        if (item is S)
        {
            this.source.Insert(index, (S)item);
        }
        else
        {
            throw new Exception("Invalid type exception");
        }
    }
}

```

包装中的 Insert 方法有一个问题。它将 D 当作参数，但是它必须将 D 插入到 IList<S> 中。由于 D 是 S 的基类型，不是所有的 D 都是 S，因此 Insert 操作可能会失败。此示例与数组的变化有相似之处。当将对象插入 object[] 时，将执行动态类型检查，因为 object[] 在运行时可能实际为 string[]。例如：

C#

```

object[] objects = new string[10];

// no problem, adding a string to a string[]
objects[0] = "hello";

// runtime exception, adding an object to a string[]
objects[1] = new object();

```

在 IList<> 示例中，当实际类型在运行时与需要的类型不匹配时，可以仅仅引发 Insert 方法的包装。所以，您同样可以想象得到编译器将为程序员自动生成此包装。然而，有时候并不应该执行此策略。IndexOf 方法在集合中搜索所提供的项，如果找到该项，则返回该项在集合中的索引。然而，如果没有找到该项，IndexOf 方法将仅仅返回 -1，而并不引发。这种类型的包装不能由自动生成的包装提供。

到目前为止，我们描述了泛型变化问题的两种最简单的解决方法。然而，变化问题可能变得要多复杂就有多复杂。例如，当您将 List<IEnumerable<int>> 当作 List<IEnumerable<object>>，或将 List<IEnumerable<IEnumerable<int>>> 当作 List<IEnumerable<IEnumerable<object>>> 时。

生成这些包装以解决代码中的变化问题可能给代码带来巨大的系统开销。同时，它还会带来引用标识问题，因为每个包装的标识都与原始集合的标识不一样，从而会导致很微妙的 Bug。当使用泛型时，应选择类型实例化，以减少紧密关联的组件之间的不匹配问题。这可能要求在设计代码时做出一些妥协。与往常一样，设计程序时必须权衡相互冲突的要求，在设计过程中应当考虑语言中类型系统具有的约束。

有的类型系统将泛型变化作为语言的首要任务。Eiffel 是其中一个主要示例。然而，将泛型变化作为类型系统的首要任务会急剧增加 C# 的类型系统的复杂性，即使在不涉及变化的相对简单方案中也是如此。因此，C# 的设计人员觉得不包括变化才是 C# 的正确选择。

下面是上述示例的完整源代码。

C#

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;

static class VarianceWorkaround
{
    // Simple workaround for single method
    // Variance in one direction only
    public static void Add<S, D>(List<S> source, List<D> destination)
        where S : D
    {

```

```
        foreach (S sourceElement in source)
    {
        destination.Add(sourceElement);
    }
}

// Workaround for interface
// Variance in one direction only so type expressinos are natural
public static IEnumerable<D> Convert<S, D>(IEnumerable<S> source)
    where S : D
{
    return new EnumerableWrapper<S, D>(source);
}

private class EnumerableWrapper<S, D> : IEnumerable<D>
    where S : D
{
    public EnumerableWrapper(IEnumerable<S> source)
    {
        this.source = source;
    }

    public IEnumerator<D> GetEnumerator()
    {
        return new EnumeratorWrapper(this.source.GetEnumerator());
    }

    IEnumerator System.Collections.IEnumerable.GetEnumerator()
    {
        return this.GetEnumerator();
    }

    private class EnumeratorWrapper : IEnumerator<D>
    {
        public EnumeratorWrapper(IEnumerator<S> source)
        {
            this.source = source;
        }

        private Ienumerator<S> source;

        public D Current
        {
            get { return this.source.Current; }
        }

        public void Dispose()
        {
            this.source.Dispose();
        }

        object IEnumerator.Current
        {
            get { return this.source.Current; }
        }

        public bool MoveNext()
        {
            return this.source.MoveNext();
        }

        public void Reset()
        {
            this.source.Reset();
        }
    }
}

private IEnumerable<S> source;
```

```

}

// Workaround for interface
// Variance in both directions, causes issues
// similar to existing array variance
public static ICollection<D> Convert<S, D>(ICollection<S> source)
    where S : D
{
    return new CollectionWrapper<S, D>(source);
}

private class CollectionWrapper<S, D>
    : EnumerableWrapper<S, D>, ICollection<D>
    where S : D
{
    public CollectionWrapper(ICollection<S> source)
        : base(source)
    {
    }

    // variance going the wrong way ...
    // ... can yield exceptions at runtime
    public void Add(D item)
    {
        if (item is S)
        {
            this.source.Add((S)item);
        }
        else
        {
            throw new Exception(@"Type mismatch exception, due to type hole introduced
by variance.");
        }
    }

    public void Clear()
    {
        this.source.Clear();
    }

    // variance going the wrong way ...
    // ... but the semantics of the method yields reasonable semantics
    public bool Contains(D item)
    {
        if (item is S)
        {
            return this.source.Contains((S)item);
        }
        else
        {
            return false;
        }
    }

    // variance going the right way ...
    public void CopyTo(D[] array, int arrayIndex)
    {
        foreach (S src in this.source)
        {
            array[arrayIndex++] = src;
        }
    }

    public int Count
    {
        get { return this.source.Count; }
    }
}

```

```

public bool IsReadOnly
{
    get { return this.source.IsReadOnly; }
}

// variance going the wrong way ...
// ... but the semantics of the method yields reasonable semantics
public bool Remove(D item)
{
    if (item is S)
    {
        return this.source.Remove((S)item);
    }
    else
    {
        return false;
    }
}

private ICollection<S> source;
}

// Workaround for interface
// Variance in both directions, causes issues similar to existing array variance
public static IList<D> Convert<S, D>(IList<S> source)
    where S : D
{
    return new ListWrapper<S, D>(source);
}

private class ListWrapper<S, D> : CollectionWrapper<S, D>, IList<D>
    where S : D
{
    public ListWrapper(IList<S> source) : base(source)
    {
        this.source = source;
    }

    public int IndexOf(D item)
    {
        if (item is S)
        {
            return this.source.IndexOf((S)item);
        }
        else
        {
            return -1;
        }
    }

    // variance the wrong way ...
    // ... can throw exceptions at runtime
    public void Insert(int index, D item)
    {
        if (item is S)
        {
            this.source.Insert(index, (S)item);
        }
        else
        {
            throw new Exception("Invalid type exception");
        }
    }

    public void RemoveAt(int index)
    {
        this.source.RemoveAt(index);
    }
}

```

```

        }

    public D this[int index]
    {
        get
        {
            return this.source[index];
        }
        set
        {
            if (value is S)
                this.source[index] = (S)value;
            else
                throw new Exception("Invalid type exception.");
        }
    }

    private IList<S> source;
}

}

namespace GenericVariance
{
    class Program
    {
        static void PrintObjects(IEnumerable<object> objects)
        {
            foreach (object o in objects)
            {
                Console.WriteLine(o);
            }
        }

        static void AddToObjects(IList<object> objects)
        {
            // this will fail if the collection provided is a wrapped collection
            objects.Add(new object());
        }
        static void Main(string[] args)
        {
            List<int> ints = new List<int>();
            ints.Add(1);
            ints.Add(10);
            ints.Add(42);
            List<object> objects = new List<object>();

            // doesnt compile ints is not a IEnumerable<object>
            //objects.AddRange(ints);

            // does compile
            VarianceWorkaround.Add<int, object>(ints, objects);

            // would like to do this, but cant ...
            // ... ints is not an IEnumerable<object>
            //PrintObjects(ints);

            PrintObjects(VarianceWorkaround.Convert<int, object>(ints));

            AddToObjects(objects); // this works fine
            AddToObjects(VarianceWorkaround.Convert<int, object>(ints));
        }
        static void ArrayExample()
        {
            object[] objects = new string[10];

            // no problem, adding a string to a string[]
            objects[0] = "hello";
        }
    }
}

```

```
// runtime exception, adding an object to a string[]
objects[1] = new object();
}
}
```

请参见

概念

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

迭代器(C# 编程指南)

迭代器是 C# 2.0 中的新功能。迭代器是方法、`get` 访问器或运算符，它使您能够在类或结构中支持 `foreach` 迭代，而不必实现整个 `IEnumerable` 接口。您只需提供一个迭代器，即可遍历类中的数据结构。当编译器检测到迭代器时，它将自动生成 `IEnumerable` 或 `IEnumerable<T>` 接口的 `Current`、`MoveNext` 和 `Dispose` 方法。

迭代器概述

- 迭代器是可以返回相同类型的值的有序序列的一段代码。
- 迭代器可用作方法、运算符或 `get` 访问器的代码体。
- 迭代器代码使用 `yield return` 语句依次返回每个元素。`yield break` 将终止迭代。有关更多信息，请参见 [yield](#)。
- 可以在类中实现多个迭代器。每个迭代器都必须像任何类成员一样有唯一的名称，并且可以在 `foreach` 语句中被客户端代码调用，如下所示：`foreach(int x in SampleClass.Iterator2 {})`
- 迭代器的返回类型必须为 `IEnumerable`、`IEnumerator`、`IEnumerable<T>` 或 `IEnumerator<T>`。

`yield` 关键字用于指定返回的值。到达 `yield return` 语句时，会保存当前位置。下次调用迭代器时将从此位置重新开始执行。

迭代器对集合类特别有用，它提供一种简单的方法来迭代不常用的数据结构（如二进制树）。

相关章节

更多信息：

- [使用迭代器\(C# 编程指南\)](#)
- [如何：为整数列表创建迭代器块\(C# 编程指南\)](#)
- [如何：为泛型列表创建迭代器块\(C# 编程指南\)](#)
- [泛型接口\(C# 编程指南\)](#)

示例

在本示例中，`DaysOfTheWeek` 类是将一周中的各天作为字符串进行存储的简单集合类。`foreach` 循环每迭代一次，都返回集合中的下一个字符串。

C#

```
public class DaysOfTheWeek : System.Collections.IEnumerable
{
    string[] m_Days = { "Sun", "Mon", "Tue", "Wed", "Thr", "Fri", "Sat" };

    public System.Collections.IEnumerator GetEnumerator()
    {
        for (int i = 0; i < m_Days.Length; i++)
        {
            yield return m_Days[i];
        }
    }
}

class TestDaysOfTheWeek
{
    static void Main()
    {
        // Create an instance of the collection class
        DaysOfTheWeek week = new DaysOfTheWeek();

        // Iterate with foreach
        foreach (string day in week)
        {
            System.Console.Write(day + " ");
        }
    }
}
```

}

输出

Sun Mon Tue Wed Thr Fri Sat

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 22 迭代器

请参见

概念

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

使用迭代器(C# 编程指南)

创建迭代器最常用的方法是对 `IEnumerable` 接口实现 `GetEnumerator` 方法，例如：

C#

```
public System.Collections.IEnumerator GetEnumerator()
{
    for (int i = 0; i < max; i++)
    {
        yield return i;
    }
}
```

`GetEnumerator` 方法的存在使得类型成为可枚举的类型，并允许使用 `foreach` 语句。如果上面的方法是 `ListClass` 的类定义的一部分，则可以对该类使用 `foreach`，如下所示：

C#

```
static void Main()
{
    ListClass listClass1 = new ListClass();

    foreach (int i in listClass1)
    {
        System.Console.WriteLine(i);
    }
}
```

`foreach` 语句调用 `ListClass.GetEnumerator()` 并使用返回的枚举数来循环访问值。有关如何创建返回 `IEnumerator` 接口的泛型迭代器的示例，请参见[如何：为泛型列表创建迭代器块\(C# 编程指南\)](#)。

还可以使用命名的迭代器以支持通过不同的方式循环访问同一数据集合。例如，您可以提供一个按升序返回元素的迭代器，而提供按降序返回元素的另一个迭代器。迭代器还可以带有参数，以便允许客户端控制全部或部分迭代行为。下面的迭代器使用命名的迭代器 `SampleIterator` 实现 `IEnumerable` 接口：

C#

```
// Implementing the enumerable pattern
public System.Collections.IEnumerable SampleIterator(int start, int end)
{
    for (int i = start; i <= end; i++)
    {
        yield return i;
    }
}
```

命名的迭代器的调用方法如下：

C#

```
ListClass test = new ListClass();
foreach (int n in test.SampleIterator(1, 10))
{
    System.Console.WriteLine(n);
}
```

可以在同一个迭代器中使用多个 `yield` 语句，如下面的示例所示：

C#

```
public System.Collections.IEnumerator GetEnumerator()
{
    yield return "With an iterator, ";
    yield return "more than one ";
    yield return "value can be returned";
    yield return ".";
}
```

然后可以使用下面的 **foreach** 语句输出结果：

C#

```
foreach (string element in new TestClass())
{
    System.Console.WriteLine(element);
}
```

此示例显示以下文本：

With an iterator, more than one value can be returned.

在 **foreach** 循环的每次后续迭代(或对 `IEnumerator.MoveNext` 的直接调用)中，下一个迭代器代码体将从前一个 **yield** 语句之后开始，并继续下一个语句直至到达迭代器体的结尾或遇到 **yield break** 语句。

请参见

任务

[如何：为整数列表创建迭代器块\(C# 编程指南\)](#)

[如何：为泛型列表创建迭代器块\(C# 编程指南\)](#)

参考

[yield\(C# 参考\)](#)

[对数组使用 foreach\(C# 编程指南\)](#)

[foreach, in\(C# 参考\)](#)

概念

[C# 编程指南](#)

如何：为整数列表创建迭代器块 (C# 编程指南)

此示例使用一个整数数组来构建列表 SampleCollection。`for` 循环会循环访问该集合，并生成每个项的值。然后使用 `foreach` 循环来显示该集合的项。

示例

C#

```
// Declare the collection:
public class SampleCollection
{
    public int[] items;

    public SampleCollection()
    {
        items = new int[5] { 5, 4, 7, 9, 3 };
    }

    public System.Collections.IEnumerable BuildCollection()
    {
        for (int i = 0; i < items.Length; i++)
        {
            yield return items[i];
        }
    }
}

class MainClass
{
    static void Main()
    {
        SampleCollection col = new SampleCollection();

        // Display the collection items:
        System.Console.WriteLine("Values in the collection are:");
        foreach (int i in col.BuildCollection())
        {
            System.Console.Write(i + " ");
        }
    }
}
```

输出

```
Values in the collection are:
5 4 7 9 3
```

请参见

任务

[如何：为泛型列表创建迭代器块 \(C# 编程指南\)](#)

参考

[使用迭代器 \(C# 编程指南\)](#)

[yield \(C# 参考\)](#)

[Array](#)

概念

[C# 编程指南](#)

[迭代器 \(C# 编程指南\)](#)

如何：为泛型列表创建迭代器块(C# 编程指南)

在本示例中，泛型类 Stack<T> 实现泛型接口 IEnumarator<T>。声明了一个类型 T 的数组，并使用 Push 方法给数组赋值。在 GetEnumerator 方法中，使用 **yield return** 语句返回数组的值。

还实现非泛型 GetEnumerator，因为 IEnumarable<T> 继承自 IEnumarable。此示例显示了典型的实现，在该实现中，非泛型方法直接将调用转给泛型方法。

示例

C#

```

using System.Collections;
using System.Collections.Generic;

namespace GenericIteratorExample
{
    public class Stack<T> : IEnumarable<T>
    {
        private T[] values = new T[100];
        private int top = 0;

        public void Push(T t) { values[top++] = t; }
        public T Pop() { return values[--top]; }

        // These make Stack<T> implement IEnumarable<T> allowing
        // a stack to be used in a foreach statement.
        public IEnumarator<T> GetEnumerator()
        {
            for (int i = top; --i >= 0; )
            {
                yield return values[i];
            }
        }

        IEnumarator IEnumarable.GetEnumerator()
        {
            return GetEnumerator();
        }

        // Iterate from top to bottom.
        public IEnumarable<T> TopToBottom
        {
            get
            {
                // Since we implement IEnumarable<T>
                // and the default iteration is top to bottom,
                // just return the object.
                return this;
            }
        }

        // Iterate from bottom to top.
        public IEnumarable<T> BottomToTop
        {
            get
            {
                for (int i = 0; i < top; i++)
                {
                    yield return values[i];
                }
            }
        }

        // A parameterized iterator that return n items from the top
        public IEnumarable<T> TopN(int n)
        {
    
```

```

        // in this example we return less than N if necessary
        int j = n >= top ? 0 : top - n;

        for (int i = top; --i >= j; )
        {
            yield return values[i];
        }
    }

    //This code uses a stack and the TopToBottom and BottomToTop properties
    //to enumerate the elements of the stack.
    class Test
    {
        static void Main()
        {
            Stack<int> s = new Stack<int>();
            for (int i = 0; i < 10; i++)
            {
                s.Push(i);
            }

            // Prints: 9 8 7 6 5 4 3 2 1 0
            // Foreach legal since s implements IEnumerable<int>
            foreach (int n in s)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 9 8 7 6 5 4 3 2 1 0
            // Foreach legal since s.TopToBottom returns IEnumerable<int>
            foreach (int n in s.TopToBottom)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 0 1 2 3 4 5 6 7 8 9
            // Foreach legal since s.BottomToTop returns IEnumerable<int>
            foreach (int n in s.BottomToTop)
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();

            // Prints: 9 8 7 6 5 4 3
            // Foreach legal since s.TopN returns IEnumerable<int>
            foreach (int n in s.TopN(7))
            {
                System.Console.Write("{0} ", n);
            }
            System.Console.WriteLine();
        }
    }
}

```

输出

```

9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3

```

请参见

任务

[如何：为整数列表创建迭代器块\(C# 编程指南\)](#)

[参考](#)

[使用迭代器\(C# 编程指南\)](#)

[System.Collections.Generic](#)

[IEnumerable](#)

[概念](#)

[C# 编程指南](#)

[迭代器\(C# 编程指南\)](#)

命名空间 (C# 编程指南)

使用 C# 编程时，通过两种方式来大量使用命名空间。首先，.NET Framework 使用命名空间来组织它的众多类，如下所示：

C#

```
System.Console.WriteLine("Hello World!");
```

System 是一个命名空间，**Console** 是该命名空间中包含的类。如果使用 **using** 关键字，则不必使用完整的名称，如下所示：

C#

```
using System;
```

C#

```
Console.WriteLine("Hello");
Console.WriteLine("World!");
```

有关更多信息，请参见主题 [using 指令 \(C# 参考\)](#)。

其次，在较大的编程项目中，声明自己的命名空间可以帮助控制类名称和方法名称的范围。使用 **namespace** 关键字可声明命名空间，如下例所示：

C#

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
                "SampleMethod inside SampleNamespace");
        }
    }
}
```

命名空间概述

命名空间具有以下属性：

- 组织大型代码项目。
- 以 . 运算符分隔。
- **using directive** 意味着不需要为每个类指定命名空间的名称。
- **global** 命名空间是“根”命名空间：**global::system** 始终引用 .NET Framework 命名空间 **System**。

相关章节

有关命名空间的更多信息，请参见下列主题：

- [使用命名空间 \(C# 编程指南\)](#)
- [如何：使用命名空间别名限定符 \(C# 编程指南\)](#)
- [如何：使用 My 命名空间 \(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 C# 语言规范 中的以下各章节：

- 9 命名空间

请参见

参考

[命名空间关键字\(C# 参考\)](#)

[using 指令\(C# 参考\)](#)

[:: 运算符\(C# 参考\)](#)

[. 运算符\(C# 参考\)](#)

概念

[C# 编程指南](#)

使用命名空间 (C# 编程指南)

在 C# 程序中，通过两种方式来大量使用命名空间。首先，.NET Framework 类使用命名空间来组织它的众多类。其次，在较大的编程项目中，声明自己的命名空间可以帮助控制类和方法名的范围。

访问命名空间

大多数 C# 应用程序从一个 **using** 指令节开始。该节列出应用程序将会频繁使用的命名空间，避免程序员在每次使用其中包含的方法时都要指定完全限定的名称。

例如，通过在程序开头包括行：

C#

```
using System;
```

程序员可以使用代码：

C#

```
Console.WriteLine("Hello, World!");
```

而不是：

C#

```
System.Console.WriteLine("Hello, World!");
```

命名空间别名

[using 指令 \(C# 参考\)](#)还可用于创建[命名空间](#)的别名。例如，如果使用包含嵌套命名空间的以前编写的命名空间，您可能希望声明一个别名来提供引用特定命名空间的简写方法，比如：

C#

```
using Co = Company.Proj.Nested; // define an alias to represent a namespace
```

使用命名空间来控制范围

namespace 关键字用于声明一个范围。在项目中创建范围的能力有助于组织代码，并提供创建全局唯一的类型的途径。在下面的示例中，名为 SampleClass 的类在两个命名空间中定义，其中一个命名空间嵌套在另一个之内。[. 运算符 \(C# 参考\)](#)用于区分所调用的方法。

C#

```
namespace SampleNamespace
{
    class SampleClass
    {
        public void SampleMethod()
        {
            System.Console.WriteLine(
                "SampleMethod inside SampleNamespace");
        }
    }

    // Create a nested namespace, and define another class.
    namespace NestedNamespace
    {
        class SampleClass
        {
```

```

        public void SampleMethod()
    {
        System.Console.WriteLine(
            "SampleMethod inside NestedNamespace");
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Displays "SampleMethod inside SampleNamespace."
        SampleClass outer = new SampleClass();
        outer.SampleMethod();

        // Displays "SampleMethod inside SampleNamespace."
        SampleNamespace.SampleClass outer2 = new SampleNamespace.SampleClass();
        outer2.SampleMethod();

        // Displays "SampleMethod inside NestedNamespace."
        NestedNamespace.SampleClass inner = new NestedNamespace.SampleClass();
        inner.SampleMethod();
    }
}

```

完全限定名

命名空间和类型的名称必须唯一，由指示逻辑层次结构的完全限定名描述。例如，语句 A.B 表示 A 是命名空间或类型的名称，而 B 则嵌套在其中。

下面的示例中有嵌套的类和命名空间。在每个实体的后面，需要完全限定名作为注释。

C#

```

namespace N1      // N1
{
    class C1      // N1.C1
    {
        class C2  // N1.C1.C2
        {
        }
    }
    namespace N2  // N1.N2
    {
        class C2  // N1.N2.C2
        {
        }
    }
}

```

以上代码段中：

- 命名空间 N1 是全局命名空间的成员。它的完全限定名是 N1。
- 命名空间 N2 是命名空间 N1 的成员。它的完全限定名是 N1.N2。
- 类 C1 是 N1 的成员。它的完全限定名是 N1.C1。
- 在此代码中使用了两次 C2 类名。但是，完全限定名是唯一的。第一个类名在 C1 内声明；因此其完全限定名是 N1.C1.C2。第二个类名在命名空间 N2 内声明；因此其完全限定名是 N1.N2.C2。

使用以上代码段，可以用以下方法将新的类成员 C3 添加到命名空间 N1.N2 内：

C#

```
namespace N1.N2
{
    class C3 // N1.N2.C3
    {
    }
}
```

一般情况下，应使用 :: 来引用命名空间别名或使用 global:: 来引用全局命名空间，并使用 . 来限定类型或成员。

与引用类型而不是命名空间的别名一起使用 :: 是错误的。例如：

C#

```
using Alias = System.Console;
```

C#

```
class TestClass
{
    static void Main()
    {
        // Error
        //Alias::WriteLine("Hi");

        // OK
        Alias.WriteLine("Hi");
    }
}
```

务必牢记单词 **global** 不是预定义的别名，因此 global.X 没有任何特殊的含义。仅当与 :: 一起使用时，它才获得特殊的含义。

定义名为 global 的别名将会产生警告(请参见[编译器警告\(等级 2\)CS0440](#))，因为 **global::** 始终引用全局命名空间而不是别名。例如，下面的行将产生警告：

C#

```
using global = System.Collections; // Warning
```

最好将 :: 与别名一起使用，这样可以避免意外引入其他类型。以下面的代码为例：

C#

```
using Alias = System;
```

C#

```
namespace Library
{
    public class C : Alias.Exception { }
}
```

这样做可行，但是如果接着引入一个名为 Alias 的类型，则 Alias. 将改为绑定到该类型。使用 Alias::Exception 可以确保 Alias 被当作命名空间别名，而不会被误认为类型。

有关 **global** 别名的更多细节，请参考主题[如何：使用命名空间别名限定符\(C# 编程指南\)](#)。

请参见

参考

[命名空间关键字\(C# 参考\)](#)

[. 运算符\(C# 参考\)](#)

[:: 运算符\(C# 参考\)](#)

[extern\(C# 参考\)](#)

概念

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

如何：使用命名空间别名限定符 (C# 编程指南)

当成员可能被同名的其他实体隐藏时，能够访问全局命名空间中的成员非常有用。

例如，在下面的代码中，`Console` 在 `System` 命名空间中解析为 `TestApp.Console` 而不是 `Console` 类型。

C#

```
using System;
```

C#

```
class TestApp
{
    // Define a new class called 'System' to cause problems.
    public class System { }

    // Define a constant called 'Console' to cause more problems.
    const int Console = 7;
    const int number = 66;

    static void Main()
    {
        // Error Accesses TestApp.Console
        //Console.WriteLine(number);
    }
}
```

由于类 `TestApp.System` 隐藏了 `System` 命名空间，因此使用 `System.Console` 仍然会导致错误：

C#

```
// Error Accesses TestApp.System
System.Console.WriteLine(number);
```

但是，可以通过使用 `global::System.Console` 避免这一错误，如下所示：

C#

```
// OK
global::System.Console.WriteLine(number);
```

当左侧的标识符为 **global** 时，对右侧标识符的搜索将从全局命名空间开始。例如，下面的声明将 `TestApp` 作为全局空间的一个成员进行引用。

C#

```
class TestClass : global::TestApp
```

显然，并不推荐创建自己的名为 `System` 的命名空间，您不可能遇到出现此情况的任何代码。但是，在较大的项目中，很有可能在一个窗体或其他窗体中出现命名空间重复。在这种情况下，全局命名空间限定符可保证您可以指定根命名空间。

示例

在此示例中，命名空间 `System` 用于包括类 `TestClass`，因此必须使用 `global::System.Console` 来引用 `System.Console` 类，该类被 `System` 命名空间隐藏。而且，别名 `colAlias` 用于引用命名空间 `System.Collections`；因此，将使用此别名而不是命名空间来创建 `System.Collections.Hashtable` 的实例。

C#

```
using colAlias = System.Collections;
```

```
namespace System
{
    class TestClass
    {
        static void Main()
        {
            // Searching the alias:
            colAlias::Hashtable test = new colAlias::Hashtable();

            // Add items to the table.
            test.Add("A", "1");
            test.Add("B", "2");
            test.Add("C", "3");

            foreach (string name in test.Keys)
            {
                // Seaching the gloabal namespace:
                global::System.Console.WriteLine(name + " " + test[name]);
            }
        }
    }
}
```

示例输出

A 1
B 2
C 3

请参见

参考

[. 运算符 \(C# 参考\)](#)

[:: 运算符 \(C# 参考\)](#)

[extern \(C# 参考\)](#)

概念

[C# 编程指南](#)

[命名空间 \(C# 编程指南\)](#)

如何：使用 My 命名空间 (C# 编程指南)

[Microsoft.VisualBasic.MyServices](#) 命名空间 (Visual Basic 中的 **My**) 提供对许多 .NET Framework 类的简单直观的访问，使您能够编写可与计算机、应用程序、设置、资源等交互的代码。虽然 **MyServices** 命名空间最初是为使用 Visual Basic 而设计的，但它也可以在 C# 应用程序中使用。

有关在 Visual Basic 中使用 **MyServices** 命名空间的更多信息，请参见[使用 My 开发](#)。

添加引用

在解决方案中使用 **MyServices** 类之前，必须添加一个对 Visual Basic 库的引用。

添加对 Visual Basic 库的引用

1. 在“解决方案资源管理器”中右击“引用”节点，再选择“添加引用”。
2. 出现“引用”对话框后，向下滚动列表，选择“Microsoft.VisualBasic.dll”。

您可能还需要在程序开头的 **using** 节中包括以下行。

C#

```
using Microsoft.VisualBasic.Devices;
```

示例

此示例调用 **MyServices** 命名空间中包含的各种静态方法。要编译此代码，必须在项目中添加一个对 Microsoft.VisualBasic.DLL 的引用。

C#

```
using System;
using Microsoft.VisualBasic.Devices;

class TestMyServices
{
    static void Main()
    {
        // Play a sound with the Audio class:
        Audio myAudio = new Audio();
        Console.WriteLine("Playing sound...");
        myAudio.Play(@"c:\WINDOWS\Media\chimes.wav");

        // Display time information with the Clock class:
        Clock myClock = new Clock();
        Console.Write("Current day of the week: ");
        Console.WriteLine(myClock.LocalTime.DayOfWeek);
        Console.Write("Current date and time: ");
        Console.WriteLine(myClock.LocalTime);

        // Display machine information with the Computer class:
        Computer myComputer = new Computer();
        Console.WriteLine("Computer name: " + myComputer.Name);

        if (myComputer.Network.IsAvailable)
        {
            Console.WriteLine("Computer is connected to network.");
        }
        else
        {
            Console.WriteLine("Computer is not connected to network.");
        }
    }
}
```

并不是 **MyServices** 命名空间中的所有的类都可以从 C# 应用程序调用:例如 [FileSystemProxy](#) 类就不兼容。在这种特定情况下, 可以改用作为 [FileSystem](#)(它也包含在 `VisualBasic.dll`中)的一部分的静态方法。例如, 下面介绍了如何使用这样的方法来复制目录:

C#

```
// Duplicate a directory
Microsoft.VisualBasic.FileIO.FileSystem.CopyDirectory(
    @"C:\original_directory",
    @"C:\copy_of_original_directory");
```

请参见

概念

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

[使用命名空间\(C# 编程指南\)](#)

可空类型(C# 编程指南)

可空类型是 [System.Nullable](#) 结构的实例。可空类型可以表示其基础值类型正常范围内的值，再加上一个 **null** 值。例如，`Nullable<Int32>`，读作“可空的 Int32”，可以被赋值为 -2147483648 到 2147483647 之间的任意值，也可以被赋值为 **null** 值。`Nullable<bool>` 可以被赋值为 **true** 或 **false**，或 **null**。在处理数据库和其他包含可能未赋值的元素的数据类型时，将 **null** 赋值给数值类型或布尔型的功能特别有用。例如，数据库中的布尔型字段可以存储值 **true** 或 **false**，或者，该字段也可以未定义。

C#

```
class NullableExample
{
    static void Main()
    {
        int? num = null;
        if (num.HasValue == true)
        {
            System.Console.WriteLine("num = " + num.Value);
        }
        else
        {
            System.Console.WriteLine("num = Null");
        }

        //y is set to zero
        int y = num.GetValueOrDefault();

        // num.Value throws an InvalidOperationException if num.HasValue is false
        try
        {
            y = num.Value;
        }
        catch (System.InvalidOperationException e)
        {
            System.Console.WriteLine(e.Message);
        }
    }
}
```

以上将显示输出：

```
num = Null
Nullable object must have a value.
```

可空类型概述

可空类型具有以下特性：

- 可空类型表示可被赋值为 **null** 值的值类型变量。无法创建基于引用类型的可空类型。(引用类型已支持 **null** 值。)。
- 语法 **T?** 是 `System.Nullable<T>` 的简写，此处的 **T** 为值类型。这两种形式可以互换。
- 为可空类型赋值与为一般值类型赋值的方法相同，如 `int? x = 10;` 或 `double? d = 4.108;`。
- 如果基础类型的值为 **null**，请使用 `System.Nullable.GetValueOrDefault` 属性返回该基础类型所赋的值或默认值，例如 `int j = x.GetValueOrDefault();`
- 请使用 `HasValue` 和 `Value` 只读属性测试是否为空和检索值，例如 `if(x.HasValue) j = x.Value;`
 - 如果此变量包含值，则 `HasValue` 属性返回 `True`；或者，如果此变量的值为空，则返回 `False`。
 - 如果已赋值，则 `Value` 属性返回该值，否则将引发 `System.InvalidOperationException`。
 - 可空类型变量的默认值将 `HasValue` 设置为 **false**。未定义 `Value`。
- 使用 `??` 运算符分配默认值，当前值为空的可空类型被赋值给非空类型时将应用该默认值，如 `int? x = null; int y = x ?? 10;`

?? -1;。

- 不允许使用嵌套的可空类型。将不编译下面一行:`Nullable<Nullable<int>> n;`

相关章节

有关更多信息：

- [使用可空类型\(C# 编程指南\)](#)
- [装箱可空类型\(C# 编程指南\)](#)
- [?? 运算符\(C# 参考\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [24 可空类型](#)

请参见

任务

[可空示例](#)

参考

[Nullable](#)

概念

[C# 编程指南](#)

其他资源

[Visual C#](#)

[C# 参考](#)

使用可空类型 (C# 编程指南)

可空类型可以表示基础类型的所有值，另外还可以表示 `null` 值。可空类型可通过下面两种方式中的一种声明：

```
System.Nullable<T> variable
```

- 或 -

```
T? variable
```

`T` 是可空类型的基础类型。`T` 可以是包括 `struct` 在内的任何值类型；但不能是引用类型。

有关可能使用可空类型的示例，请考虑普通的布尔变量如何能够具有两个值：`true` 和 `false`。不存在表示“未定义”的值。在很多编程应用中（最突出的是数据库交互），变量可存在于未定义的状态。例如，数据库中的某个字段可能包含值 `true` 或 `false`，但是它也可能根本不包含值。同样，可以将引用类型设置为 `null`，以指示它们未初始化。

这种不一致会导致额外的编程工作，如使用附加变量来存储状态信息、使用特殊值，等等。可空类型修饰符使 C# 能够创建表示未定义值的值类型变量。

可空类型示例

任何值类型都可用作可空类型的基础。例如：

C#

```
int? i = 10;
double? d1 = 3.14;
bool? flag = null;
char? letter = 'a';
int?[] arr = new int?[10];
```

可空类型的成员

可空类型的每个实例都具有两个公共的只读属性：

- **HasValue**

`HasValue` 属于 `bool` 类型。当变量包含非空值时，它被设置为 `true`。

- **Value**

`Value` 的类型与基础类型相同。如果 `HasValue` 为 `true`，则说明 `Value` 包含有意义的值。如果 `HasValue` 为 `false`，则访问 `Value` 将引发 `InvalidOperationException`。

在此示例中，`HasValue` 成员用于在尝试显示变量之前测试它是否包含值。

C#

```
int? x = 10;
if (x.HasValue)
{
    System.Console.WriteLine(x.Value);
}
else
{
    System.Console.WriteLine("Undefined");
}
```

也可以通过下面的方法测试是否包含值：

C#

```
int? y = 10;
if (y != null)
{
    System.Console.WriteLine(y.Value);
}
```

```
else
{
    System.Console.WriteLine("Undefined");
}
```

显式转换

可空类型可强制转换为常规类型，方法是使用强制转换来显式转换或者通过使用 `Value` 属性来转换。例如：

C#

```
int? n = null;

//int m1 = n;      // Will not compile.
int m2 = (int)n; // Compiles, but will create an exception if x is null.
int m3 = n.Value; // Compiles, but will create an exception if x is null.
```

如果两种数据类型之间定义了用户定义的转换，则同一转换也可用于这些数据类型的可空版本。

隐式转换

可使用 `null` 关键字将可空类型的变量设置为空，如下所示：

C#

```
int? n1 = null;
```

从普通类型到可空类型的转换是隐式的。

C#

```
int? n2;
n2 = 10; // Implicit conversion.
```

运算符

可空类型还可以使用预定义的一元和二元运算符，以及现有的任何用户定义的值类型运算符。如果操作数为空，这些运算符将产生一个空值；否则运算符将使用包含的值来计算结果。例如：

C#

```
int? a = 10;
int? b = null;

a++;          // Increment by 1, now a is 11.
a = a * 10;   // Multiply by 10, now a is 110.
a = a + b;    // Add b, now a is null.
```

在执行可空类型的比较时，如果其中一个可空类型为 `null`，则比较结果将始终为 `false`。因此，一定不要以为由于一个比较结果为 `false`，相反的情况就会为 `true`。例如：

C#

```
int? num1 = 10;
int? num2 = null;
if (num1 >= num2)
{
    System.Console.WriteLine("num1 is greater than or equal to num2");
}
else
{
    // num1 is NOT less than num2
}
```

上面的 **else** 语句中的结论无效，因为 `num2` 为 **null**，所以不包含值。

比较两个均为 **null** 的可空类型时结果为 **true**。

?? 运算符

?? 运算符定义在将可空类型分配给非可空类型时返回的默认值。

C#

```
int? c = null;  
  
// d = c, unless c is null, in which case d = -1.  
int d = c ?? -1;
```

此运算符还可用多个可空类型。例如：

C#

```
int? e = null;  
int? f = null;  
  
// g = e or f, unless e and f are both null, in which case g = -1.  
int g = e ?? f ?? -1;
```

bool? 类型

bool? 可空类型可以包含三个不同的值：**true**、**false** 和 **null**。它们本身不能用于条件语句，如 **if**、**for** 或 **while**。例如，下面的代码编译失败，并将报告[编译器错误 CS0266](#)：

```
bool? b = null;  
if (b) // Error CS0266.  
{  
}
```

这是不允许的，因为 **null** 在条件上下文中意味着什么并不清楚。为了能在条件语句中使用，可空布尔值可以显式强制转换为 **bool**，但是如果对象有值 **null**，将引发 **InvalidOperationException**。因此，在强制转换为 **bool** 前检查 [HasValue](#) 属性很重要。

可空布尔值类似于 SQL 中使用的布尔变量类型。为了确保 **&** 和 **|** 运算符产生的结果与 SQL 的三值布尔类型一致，提供了以下预定义的运算符：

```
bool? operator &(bool? x, bool? y)  
bool? operator |(bool? x, bool? y)
```

下表中列出了这些运算符的结果：

x	y	x&y	x y
True	True	True	True
True	False	False	True
True	Null	Null	True
False	True	False	True
False	False	False	False
False	Null	False	Null

Null	True	Null	True
Null	False	False	Null
Null	Null	Null	Null

请参见

参考

[装箱可空类型\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[可空类型\(C# 编程指南\)](#)

装箱可空类型(C# 编程指南)

基于可空类型的对象只在该对象为非空时装箱。如果 `HasValue` 为 `false`, 则不装箱, 而是将对象引用直接赋值为 `null`。例如:

```
bool? b = null;
object o = b;
// Now o is null.
```

如果对象非空, 也就是说, 如果 `HasValue` 为 `true`, 则会发生装箱过程, 但只将可空对象所基于的基础类型装箱。如果将非空的可空值类型装箱, 将使值类型本身(而不是包装该值类型的 `System.Nullable`)装箱。例如:

```
bool? b = false;
int? i = 44;
object bBoxed = b; // bBoxed contains a boxed bool.
object iBoxed = i; // iBoxed contains a boxed int.
```

对于那些通过装箱非可空类型而创建的类型来说, 两种装箱对象是完全相同的。并且, 像非可空装箱类型一样, 可以将它们取消装箱, 使其成为可空类型, 如下所示:

```
bool? b2 = (bool?)bBoxed;
int? i2 = (int?)iBoxed;
```

备注

可空类型在装箱时的行为具有两个优点:

1. 可以测试可空对象及其装箱的对应项是否为空:

```
bool? b = null;
object boxedB = b;
if (b == null)
{
    // True.
}
if (boxedB == null)
{
    // Also true.
}
```

2. 装箱的可空类型完全支持基础类型的功能:

```
double? d = 44.4;
object iBoxed = d;
// Access IConvertible interface implemented by double.
IConvertible ic = (IConvertible)iBoxed;
int i = ic.ToInt32(null);
string str = ic.ToString();
```

有关可空类型的更多示例(包括装箱行为), 请参见[可空示例](#)。

请参见

任务

[如何: 标识可空类型\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[可空类型\(C# 编程指南\)](#)

如何：标识可空类型 (C# 编程指南)

可以使用 C# `typeof` 运算符来创建表示可空类型的 `Type` 对象：

```
System.Type type = typeof(int?);
```

还可以使用 `System.Reflection` 命名空间的类和方法来生成表示可空类型的 `Type` 对象。但是，如果您试图使用 `GetType` 方法或 `is` 运算符在运行时获得可空类型变量的类型信息，得到的结果是表示基础类型而不是可空类型本身的 `Type` 对象。

如果对可空类型调用 `GetType`，则在该类型被隐式转换为 `Object` 时将执行装箱操作。因此，`GetType` 总是返回表示基础类型而不是可空类型的 `Type` 对象。

```
int? i = 5;
Type t = i.GetType();
Console.WriteLine(t.FullName); // "System.Int32"
```

C# 的 `is` 运算符还可以作用于可空的基础类型。因此，不能使用 `is` 来确定变量是否为可空类型。下面的示例演示 `is` 运算符将 `Nullable<int>` 变量视为 `int` 变量。

```
static void Main(string[] args)
{
    int? i = 5;
    if (i is int) // true
        //...
}
```

示例

使用下面的代码来确定 `Type` 对象是否表示可空类型。请记住，如果 `Type` 对象是通过调用 `GetType` 返回的，则此代码始终返回 `false`，如上所述。

```
if (type.IsGenericType && type.GetGenericTypeDefinition() == typeof(Nullable<>)) {...}
```

请参见

参考

[装箱可空类型 \(C# 编程指南\)](#)

概念

[可空类型 \(C# 编程指南\)](#)

不安全代码和指针(C# 编程指南)

为了保持类型安全，默认情况下，C# 不支持指针运算。不过，通过使用 `unsafe` 关键字，可以定义可使用指针的不安全上下文。有关指针的更多信息，请参见主题[指针类型](#)。

注意

在公共语言运行库 (CLR) 中，不安全代码是指无法验证的代码。C# 中的不安全代码不一定是危险的，只是其安全性无法由 CLR 进行验证的代码。因此，CLR 只对在完全受信任的程序集中的不安全代码执行操作。如果使用不安全代码，由您负责确保您的代码不会引起安全风险或指针错误。有关更多信息，请参见[安全性与 C#](#)。

不安全代码概述

不安全代码具有下列属性：

- 方法、类型和可被定义为不安全的代码块。
- 在某些情况下，通过移除数组界限检查，不安全代码可提高应用程序的性能。
- 当调用需要指针的本机函数时，需要使用不安全代码。
- 使用不安全代码将引起安全风险和稳定性风险。
- 在 C# 中，为了编译不安全代码，必须用 `/unsafe` 编译应用程序。

相关章节

有关更多信息，请参见：

- [指针类型\(C# 编程指南\)](#)
- [固定大小的缓冲区\(C# 编程指南\)](#)
- [如何:使用指针复制字节数组\(C# 编程指南\)](#)
- [如何:使用 Windows ReadFile 函数\(C# 编程指南\)](#)
- [unsafe\(C# 参考\)](#)
- [“不安全代码”示例](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [18 不安全代码](#)
- [B 3 不安全代码的语法扩展](#)

请参见

概念

[C# 编程指南](#)

固定大小的缓冲区(C# 编程指南)

在 C# 2.0 中，可以使用 `fixed` 语句在数据结构中创建固定大小的数组。使用现有代码（如：使用其他语言、预先存在的 DLL 或 COM 项目编写的代码）时这种方法非常有用。固定数组可采用允许普通结构成员使用的任何属性或修饰符。唯一的限制是，数组类型必须是 `bool`、`byte`、`char`、`short`、`int`、`long`、`sbyte`、`ushort`、`uint`、`ulong`、`float` 或 `double`。

```
private fixed char name[30];
```

备注

在以前版本的 C# 中，声明 C++ 样式的固定大小结构是很困难的，因为包含数组的 C# 结构不包含数组元素，而是包含对元素的引用。

C# 2.0 添加了在[结构](#)（当用在[不安全的代码块](#)中时）中嵌入固定大小的数组的功能。

例如，在 C# 2.0 之前，下面的 `struct` 的大小为 8 字节，其中 `pathName` 数组是对堆分配的数组的引用：

C#

```
public struct MyArray
{
    public char[] pathName;
    private int reserved;
}
```

在 C# 2.0 中，`struct` 可使用嵌入数组进行声明：

C#

```
public struct MyArray // This code must appear in an unsafe block
{
    public fixed char pathName[128];
}
```

在此结构中，`pathName` 数组具有固定的大小和位置，因此可用在其他不安全的代码中。

128 个元素的 `char` 数组的大小为 256 字节。在固定大小的 `char` 缓冲区中，每个字符始终占用两个字节，而与编码无关。即使将 `char` 缓冲区封送到具有 `CharSet = CharSet.Auto` 或 `CharSet = CharSet.Ansi` 的 API 方法或结构，也是如此。有关更多信息，请参见 [CharSet](#)。

另一种常见的固定大小的数组是 `bool` 数组。`bool` 数组中的元素的大小始终为一个字节。`bool` 数组不适合用于创建位数组或位缓冲区。

注意

除了用 `stackalloc` 创建的内存，C# 编译器和公共语言运行库 (CLR) 不执行任何安全缓冲区溢出检查。与所有不安全代码一样，请谨慎使用。

不安全缓冲区与常规数组有如下差异：

- 不安全缓冲区只能用在不安全上下文中。
- 不安全缓冲区始终是向量（或一维数组）。
- 数组的声明应包括计数，如 `char id[8]`。而不能使用 `char id[]`。
- 不安全缓冲区只能是不安全上下文中的结构的实例字段。

请参见

任务

[如何：使用 PInvoke 从托管代码调用本机 DLL](#)

[如何：使用 PInvoke 封送数组](#)

[如何: 使用 PInvoke 封送嵌入式指针](#)

[如何: 使用 PInvoke 封送函数指针](#)

[如何: 使用 PInvoke 封送字符串](#)

参考

[fixed 语句 \(C# 参考\)](#)

概念

[C# 编程指南](#)

[不安全代码和指针 \(C# 编程指南\)](#)

[数组的默认封送处理](#)

如何：使用 Windows ReadFile 函数 (C# 编程指南)

下面的示例通过读取并显示一个文本文件来演示 Windows **ReadFile** 函数。**ReadFile** 函数需要使用 **unsafe** 代码，因为它需要一个作为参数的指针。

传递到 **Read** 函数的字节数组是托管类型。这意味着公共语言运行库 (CLR) 垃圾回收器可能会随意地对数组使用的内存进行重新定位。为了防止出现这种情况，使用 **fixed** 来获取指向内存的指针并对它进行标记，以便垃圾回收器不会移动它。在 **fixed** 块的末尾，内存将自动返回，以便能够通过垃圾回收移动。

此功能称为“声明式锁定”。锁定的好处是系统开销非常小，除非在 **fixed** 块中发生垃圾回收(但此情况不太可能发生)。

示例

C#

```

class FileReader
{
    const uint GENERIC_READ = 0x80000000;
    const uint OPEN_EXISTING = 3;
    System.IntPtr handle;

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe System.IntPtr CreateFile(
    (
        string FileName,           // file name
        uint DesiredAccess,        // access mode
        uint ShareMode,            // share mode
        uint SecurityAttributes,   // Security Attributes
        uint CreationDisposition, // how to create
        uint FlagsAndAttributes,   // file attributes
        int hTemplateFile          // handle to template file
    );

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe bool ReadFile(
    (
        System.IntPtr hFile,       // handle to file
        void* pBuffer,             // data buffer
        int NumberOfBytesToRead,   // number of bytes to read
        int* pNumberOfBytesRead,   // number of bytes read
        int Overlapped             // overlapped buffer
    );

    [System.Runtime.InteropServices.DllImport("kernel32", SetLastError = true)]
    static extern unsafe bool CloseHandle(
    (
        System.IntPtr hObject // handle to object
    );

    public bool Open(string FileName)
    {
        // open the existing file for reading
        handle = CreateFile(
        (
            FileName,
            GENERIC_READ,
            0,
            0,
            OPEN_EXISTING,
            0,
            0
        ));

        if (handle != System.IntPtr.Zero)
        {
            return true;
        }
    }
}

```

```

        else
    {
        return false;
    }
}

public unsafe int Read(byte[] buffer, int index, int count)
{
    int n = 0;
    fixed (byte* p = buffer)
    {
        if (!ReadFile(handle, p + index, count, &n, 0))
        {
            return 0;
        }
    }
    return n;
}

public bool Close()
{
    return CloseHandle(handle);
}
}

class Test
{
    static int Main(string[] args)
    {
        if (args.Length != 1)
        {
            System.Console.WriteLine("Usage : ReadFile <FileName>");
            return 1;
        }

        if (!System.IO.File.Exists(args[0]))
        {
            System.Console.WriteLine("File " + args[0] + " not found.");
            return 1;
        }

        byte[] buffer = new byte[128];
        FileReader fr = new FileReader();

        if (fr.Open(args[0]))
        {
            // Assume that an ASCII file is being read.
            System.Text.ASCIIEncoding Encoding = new System.Text.ASCIIEncoding();

            int bytesRead;
            do
            {
                bytesRead = fr.Read(buffer, 0, buffer.Length);
                string content = Encoding.GetString(buffer, 0, bytesRead);
                System.Console.Write("{0}", content);
            }
            while (bytesRead > 0);

            fr.Close();
            return 0;
        }
        else
        {
            System.Console.WriteLine("Failed to open requested file");
            return 1;
        }
    }
}

```

[请参见](#)

[参考](#)

[指针类型\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

[垃圾回收](#)

指针类型(C# 编程指南)

在不安全的上下文中，类型可以是指针类型以及值类型或引用类型。指针类型声明具有下列形式之一：

```
type* identifier;
void* identifier; //allowed but not recommended
```

下列类型都可以是指针类型：

- `sbyte`、`byte`、`short`、`ushort`、`int`、`uint`、`long`、`ulong`、`char`、`float`、`double`、`decimal` 或 `bool`。
- 任何枚举类型。
- 任何指针类型。
- 仅包含非托管类型的字段的任何用户定义的结构类型。

指针类型不继承 `object`，并且指针类型与 `object` 之间不存在转换。此外，装箱和取消装箱不支持指针。但是，允许在不同指针类型之间以及指针类型与整型之间进行转换。

当在同一个声明中声明多个指针时，* 仅与基础类型一起使用，而不是作为每个指针名称的前缀。例如：

```
int* p1, p2, p3; // Ok
int *p1, *p2, *p3; // Invalid in C#
```

指针不能指向引用或包含引用的结构，因为即使有指针指向对象引用，该对象引用也可能会被执行垃圾回收。GC 并不注意是否有任何类型的指针指向对象。

`myType*` 类型的指针变量的值是 `myType` 类型的变量的地址。下面是指针类型声明的示例：

示例	说明
<code>int* p</code>	<code>p</code> 是指向整数的指针
<code>int** p</code>	<code>p</code> 是指向整数的指针的指针
<code>int*[] p</code>	<code>p</code> 是指向整数的指针的一维数组
<code>char* p</code>	<code>p</code> 是指向字符的指针
<code>void* p</code>	<code>p</code> 是指向未知类型的指针

指针间接寻址运算符 * 可用于访问位于指针变量所指向的位置的内容。例如，对于下面的声明，

```
int* myVariable;
```

表达式 `*myVariable` 表示在 `myVariable` 中包含的地址处找到的 `int` 变量。

不能对 `void*` 类型的指针应用间接寻址运算符。但是，可以使用强制转换将 `void` 指针转换为其他指针类型，反之亦然。

指针可以为 `null`。如果将间接寻址运算符应用于 `null` 指针，则会导致由实现定义的行为。

注意，在方法之间传递指针会导致未定义的行为。示例包括通过 `Out` 或 `Ref` 参数向局部变量返回指针或作为函数结果向局部变量返回指针。如果将指针设置在固定的块中，它所指向的变量可能不再是固定的。

下表列出可在不安全的上下文中针对指针执行的运算符和语句：

运算符/语句	用途
*	执行指针间接寻址。

->	通过指针访问结构的成员。
[]	对指针建立索引。
&	获取变量的地址。
++ 和 --	递增或递减指针。
加、减	执行指针算法。
==、!=、<、>、<= 和 >=	比较指针。
stackalloc	在堆栈上分配内存。
fixed 语句	临时固定变量以便可以找到其地址。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 18 不安全代码

请参见

参考

[指针转换\(C# 编程指南\)](#)

[指针表达式\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

[装箱和取消装箱\(C# 编程指南\)](#)

其他资源

[类型\(C# 参考\)](#)

指针转换(C# 编程指南)

下表显示了预定义的隐式指针转换。隐式转换可能在多种情形下发生，包括调用方法时和在赋值语句中。

隐式指针转换

从	到
任何指针类型	void*
null	任何指针类型

显式指针转换用于在不存在隐式转换时通过使用强制转换表达式来执行转换。下表显示了这些转换。

显式指针转换

从	到
任何指针类型	所有其他指针类型
sbyte、byte、short、ushort、int、uint、long 或 ulong	任何指针类型
任何指针类型	sbyte、byte、short、ushort、int、uint、long 或 ulong

示例

在下面的示例中，一个指向 **int** 的指针被转换为指向 **byte** 的指针。注意，该指针指向变量的最低地址字节。连续递增该结果直到到达 **int** 的大小(4 字节)，即可显示变量的剩余字节。

C#

```
// compile with: /unsafe
```

C#

```
class ClassConvert
{
    static void Main()
    {
        int number = 1024;

        unsafe
        {
            // Convert to byte:
            byte* p = (byte*)&number;

            System.Console.Write("The 4 bytes of the integer:");

            // Display the 4 bytes of the int variable:
            for (int i = 0 ; i < sizeof(int) ; ++i)
            {
                System.Console.Write(" {0:X2}", *p);
                // Increment the pointer:
                p++;
            }
            System.Console.WriteLine();
            System.Console.WriteLine("The value of the integer: {0}", number);
        }
    }
}
```

输出

The 4 bytes of the integer: 00 04 00 00

```
The value of the integer: 1024
```

请参见

参考

[指针表达式\(C# 编程指南\)](#)

[指针类型\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[类型\(C# 参考\)](#)

指针表达式 (C# 编程指南)

本节讨论下列指针表达式：

[获取变量的值](#)

[获取变量的地址](#)

[如何：通过指针访问成员 \(C# 编程指南\)](#)

[如何：通过指针访问数组元素 \(C# 编程指南\)](#)

[操作指针](#)

[请参见](#)

[参考](#)

[指针转换 \(C# 编程指南\)](#)

[指针类型 \(C# 编程指南\)](#)

[unsafe \(C# 参考\)](#)

[fixed 语句 \(C# 参考\)](#)

[stackalloc \(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[类型 \(C# 参考\)](#)

如何: 获取指针变量的值(C# 编程指南)

使用指针间接运算符可获取位于指针所指向的位置的变量。表达式采用下面的形式，其中，`p` 是指针类型：

```
*p;
```

不能对除指针类型以外的任何类型的表达式使用一元间接寻址运算符。此外，不能将它应用于 `void` 指针。

当向 `null` 指针应用间接寻址运算符时，结果将取决于具体的实现。

示例

下面的示例使用不同类型的指针访问 `char` 类型的变量。注意，`theChar` 的地址在不同的运行中是不同的，因为分配给变量的物理地址可能会更改。

C#

```
// compile with: /unsafe
```

C#

```
unsafe class TestClass
{
    static void Main()
    {
        char theChar = 'Z';
        char* pChar = &theChar;
        void* pVoid = pChar;
        int* pInt = (int*)pVoid;

        System.Console.WriteLine("Value of theChar = {0}", theChar);
        System.Console.WriteLine("Address of theChar = {0:X2}", (int)pChar);
        System.Console.WriteLine("Value of pChar = {0}", *pChar);
        System.Console.WriteLine("Value of pInt = {0}", *pInt);
    }
}
```

示例输出

```
Value of theChar = Z
Address of theChar = 12F718
Value of pChar = Z
Value of pInt = 90
```

请参见

参考

[指针表达式\(C# 编程指南\)](#)

[指针类型\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[类型\(C# 参考\)](#)

如何: 获取变量的地址 (C# 编程指南)

要获取计算结果为固定变量的一元表达式的地址, 请使用 address-of 运算符:

```
int number;
int* p = &number; //address-of operator &
```

address-of 运算符仅适用于变量。如果该变量是可移动变量, 则在获取其地址之前, 可以使用 [fixed 语句](#) 暂时固定此变量。

确保初始化该变量是程序员的责任。如果变量未初始化, 编译器不会发出错误消息。

不能获取常数或值的地址。

示例

此示例声明一个指向 `int` 的指针 `p`, 并将整数变量 `number` 的地址赋值给该指针。变量 `number` 被初始化为对 `*p` 赋值的结果。注释掉此赋值语句将取消对变量 `number` 的初始化, 但是不会发出编译时错误。注意该示例如何使用 [成员访问](#) 运算符 `->` 来获取和显示存储在指针中的地址。

C#

```
// compile with: /unsafe
```

C#

```
class AddressOfOperator
{
    static void Main()
    {
        int number;

        unsafe
        {
            // Assign the address of number to a pointer:
            int* p = &number;

            // Commenting the following statement will remove the
            // initialization of number.
            *p = 0xffff;

            // Print the value of *p:
            System.Console.WriteLine("Value at the location pointed to by p: {0:X}", *p);

            // Print the address stored in p:
            System.Console.WriteLine("The address stored in p: {0}", p->ToString());
        }

        // Print the value of the variable number:
        System.Console.WriteLine("Value of the variable number: {0:X}", number);
    }
}
```

示例输出

```
Value at the location pointed to by p: FFFF
The address stored in p: 65535
Value of the variable number: FFFF
```

请参见

参考

- [指针表达式 \(C# 编程指南\)](#)
- [指针类型 \(C# 编程指南\)](#)
- [unsafe \(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[类型\(C# 参考\)](#)

如何：通过指针访问成员 (C# 编程指南)

要访问在不安全的上下文中声明的结构的成员，您可以使用以下示例中所示的成员访问运算符，其中，`p` 是指向包含成员 `x` 的结构的指针。

```
CoOrds* p = &home;
p->x = 25; //member access operator ->
```

示例

此示例声明并实例化了包含两个坐标 (`x` 和 `y`) 的结构 `CoOrds`。此示例通过使用成员访问运算符 `->` 和一个指向实例 `home` 的指针为 `x` 和 `y` 赋值。

注意

请注意，表达式 `p->x` 等效于表达式 `(*p).x`，使用这两个表达式可获得相同的结果。

C#

```
// compile with: /unsafe
```

C#

```
struct CoOrds
{
    public int x;
    public int y;
}

class AccessMembers
{
    static void Main()
    {
        CoOrds home;

        unsafe
        {
            CoOrds* p = &home;
            p->x = 25;
            p->y = 12;

            System.Console.WriteLine("The coordinates are: x={0}, y={1}", p->x, p->y );
        }
    }
}
```

请参见

参考

[指针表达式 \(C# 编程指南\)](#)

[指针类型 \(C# 编程指南\)](#)

[unsafe \(C# 参考\)](#)

[fixed 语句 \(C# 参考\)](#)

[stackalloc \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[类型 \(C# 参考\)](#)

如何：通过指针访问数组元素(C# 编程指南)

在不安全的上下文中，可通过使用指针元素访问来访问内存中的元素，如下面的示例所示：

```
char* charPointer = stackalloc char[123];
for (int i = 65; i < 123; i++)
{
    charPointer[i] = (char)i; //access array elements
}
```

方括号中的表达式必须能够隐式转换为 **int**、**uint**、**long** 或 **ulong**。操作 `p[e]` 等效于 `*(p+e)`。与 C 和 C++ 一样，指针元素访问不检查越界错误。

示例

在此示例中，123 内存位置被分配给字符数组 `charPointer`。该数组用于在两个 `for` 循环中显示小写字母和大写字母。

请注意，表达式 `charPointer[i]` 等效于表达式 `*(charPointer + i)`，使用这两个表达式可获得相同的结果。

C#

```
// compile with: /unsafe
```

C#

```
class Pointers
{
    unsafe static void Main()
    {
        char* charPointer = stackalloc char[123];

        for (int i = 65; i < 123; i++)
        {
            charPointer[i] = (char)i;

            // Print uppercase letters:
            System.Console.WriteLine("Uppercase letters:");
            for (int i = 65; i < 91; i++)
            {
                System.Console.Write(charPointer[i]);
            }
            System.Console.WriteLine();

            // Print lowercase letters:
            System.Console.WriteLine("Lowercase letters:");
            for (int i = 97; i < 123; i++)
            {
                System.Console.Write(charPointer[i]);
            }
        }
    }
}
```

示例输出

```
Uppercase letters:
ABCDEFGHIJKLMNOPQRSTUVWXYZ
Lowercase letters:
abcdefghijklmnopqrstuvwxyz
```

请参见

参考

[指针表达式\(C# 编程指南\)](#)

[指针类型\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[类型\(C# 参考\)](#)

操作指针(C# 编程指南)

本章节包括下列指针操作：

[增加和减少](#)

[算术运算](#)

[比较](#)

[请参见](#)

[参考](#)

[指针表达式\(C# 编程指南\)](#)

[C# 运算符](#)

[指针类型\(C# 编程指南\)](#)

[/unsafe\(启用不安全模式\)\(C# 编译器选项\)](#)

[概念](#)

[C# 编程指南](#)

如何：递增和递减指针 (C# 编程指南)

使用增量和减量运算符 `++` 和 `--` 可以将 `pointer-type*` 类型的指针的位置改变 `sizeof(pointer-type)`。增量和减量表达式的形式如下：

```
++p;
P++;
--p;
p--;
```

增量和减量运算符可应用于除 `void*` 类型以外的任何类型的指针。

对 `pointer-type` 类型的指针应用增量运算符的效果是将指针变量中包含的地址增加 `sizeof(pointer-type)`。

对 `pointer-type` 类型的指针应用减量运算符的效果是从指针变量中包含的地址减去 `sizeof(pointer-type)`。

当运算溢出指针范围时，不会产生异常，实际结果取决于具体实现。

示例

此示例通过将指针增加 `int` 的大小来遍历一个数组。对于每一步，此示例都显示数组元素的地址和内容。

C#

```
// compile with: /unsafe
```

C#

```
class IncrDecr
{
    unsafe static void Main()
    {
        int[] numbers = {0,1,2,3,4};

        // Assign the array address to the pointer:
        fixed (int* p1 = numbers)
        {
            // Step through the array elements:
            for(int* p2=p1; p2<p1+numbers.Length; p2++)
            {
                System.Console.WriteLine("Value:{0} @ Address:{1}", *p2, (long)p2);
            }
        }
    }
}
```

示例输出

```
Value:0 @ Address:12860272
Value:1 @ Address:12860276
Value:2 @ Address:12860280
Value:3 @ Address:12860284
Value:4 @ Address:12860288
```

请参见

参考

[指针表达式 \(C# 编程指南\)](#)

[C# 运算符](#)

[操作指针 \(C# 编程指南\)](#)

[指针类型 \(C# 编程指南\)](#)

[unsafe \(C# 参考\)](#)

[fixed 语句 \(C# 参考\)](#)

[stackalloc \(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[类型\(C# 参考\)](#)

指针的算术运算(C# 编程指南)

本主题讨论使用算术运算符 + 和 - 来操作指针。

注意

不能对 void 指针执行任何算术运算。

对指针执行加减数值操作

可以将类型为 `int`、`uint`、`long` 或 `ulong` 的值 `n` 与 `void*` 以外任何类型的指针 `p` 相加。结果 `p+n` 是加上 `n * sizeof(p)` to the address of `p` 得到的指针。同样, `p-n` 是从 `p` 的地址中减去 `n * sizeof(p)` 得到的指针。

指针相减

也可以对相同类型的指针进行减法运算。计算结果的类型始终为 `long`。例如, 如果 `p1` 和 `p2` 都是类型为 `pointer-type*` 的指针, 则表达式 `p1-p2` 的计算结果为:

```
((long)p1 - (long)p2)/sizeof(pointer_type)
```

当算术运算溢出指针范围时, 不会产生异常, 并且结果取决于具体实现。

示例

C#

```
// compile with: /unsafe
```

C#

```
class PointerArithmetic
{
    unsafe static void Main()
    {
        int* memory = stackalloc int[30];
        long* difference;
        int* p1 = &memory[4];
        int* p2 = &memory[10];

        difference = (long*)(p2 - p1);

        System.Console.WriteLine("The difference is: {0}", (long)difference);
    }
}
```

输出

The difference is: 6

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 18.5.6 指针算法

请参见

参考

[指针表达式\(C# 编程指南\)](#)

[C# 运算符](#)

[操作指针\(C# 编程指南\)](#)

[指针类型\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

其他资源

[类型\(C# 参考\)](#)

指针比较(C# 编程指南)

可应用下面的运算符比较任意类型的指针：

`== != < > <= >=`

比较运算符比较两个操作数的地址，就像他们是无符号整数一样。

示例

C#

```
// compile with: /unsafe
```

C#

```
class CompareOperators
{
    unsafe static void Main()
    {
        int x = 234;
        int y = 236;
        int* p1 = &x;
        int* p2 = &y;

        System.Console.WriteLine(p1 < p2);
        System.Console.WriteLine(p2 < p1);
    }
}
```

示例输出

True

False

请参见

参考

[指针表达式\(C# 编程指南\)](#)

[C# 运算符](#)

[操作指针\(C# 编程指南\)](#)

[指针类型\(C# 编程指南\)](#)

[unsafe\(C# 参考\)](#)

[fixed 语句\(C# 参考\)](#)

[stackalloc\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[类型\(C# 参考\)](#)

如何：使用指针复制字节数组 (C# 编程指南)

下面的示例使用指针将字节从一个数组复制到另一个使用指针的数组。

此示例使用 `unsafe` 关键字，它允许在 `Copy` 方法内使用指针。`fixed` 语句用于声明指向源数组和目标数组的指针。这将锁定源数组和目标数组在内存中的位置，使其不会因为垃圾回收操作而移动。这些内存块将在 `fixed` 块结束时取消锁定。因为本示例中 `Copy` 函数使用了 `unsafe` 关键字，它必须使用 `/unsafe` 编译器选项进行编译。

示例

C#

```
// compile with: /unsafe
```

C#

```
class TestCopy
{
    // The unsafe keyword allows pointers to be used within the following method:
    static unsafe void Copy(byte[] src, int srcIndex, byte[] dst, int dstIndex, int count)
    {
        if (src == null || srcIndex < 0 ||
            dst == null || dstIndex < 0 || count < 0)
        {
            throw new System.ArgumentException();
        }

        int srcLen = src.Length;
        int dstLen = dst.Length;
        if (srcLen - srcIndex < count || dstLen - dstIndex < count)
        {
            throw new System.ArgumentException();
        }

        // The following fixed statement pins the location of the src and dst objects
        // in memory so that they will not be moved by garbage collection.
        fixed (byte* pSrc = src, pDst = dst)
        {
            byte* ps = pSrc;
            byte* pd = pDst;

            // Loop over the count in blocks of 4 bytes, copying an integer (4 bytes) at a
            time:
            for (int i = 0 ; i < count / 4 ; i++)
            {
                *((int*)pd) = *((int*)ps);
                pd += 4;
                ps += 4;
            }

            // Complete the copy by moving any bytes that weren't moved in blocks of 4:
            for (int i = 0; i < count % 4 ; i++)
            {
                *pd = *ps;
                pd++;
                ps++;
            }
        }
    }

    static void Main()
    {
        byte[] a = new byte[100];
        byte[] b = new byte[100];
```

```
for (int i = 0; i < 100; ++i)
{
    a[i] = (byte)i;
}

Copy(a, 0, b, 0, 100);
System.Console.WriteLine("The first 10 elements are:");

for (int i = 0; i < 10; ++i)
{
    System.Console.Write(b[i] + " ");
}
System.Console.WriteLine("\n");
}
```

输出

The first 10 elements are:
0 1 2 3 4 5 6 7 8 9

请参见

任务

[“不安全代码”示例](#)

参考

[/unsafe\(启用不安全模式\)\(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

其他资源

[垃圾回收](#)

XML 文档注释(C# 编程指南)

在 Visual C# 中，可以为代码创建文档，方法是在 XML 标记所指的代码块前面，直接在源代码的特殊注释字段中包括 XML 标记。例如：

```
/// <summary>
/// This class performs an important function.
/// </summary>
public class MyClass{}
```

使用 [/doc](#) 进行编译时，编译器将在源代码中搜索所有的 XML 标记，并创建一个 XML 文档文件。

注意

XML doc 注释不是元数据；它们不包括在编译的程序集中，因此无法通过反射对其进行访问。

本节内容

- [建议的文档注释标记](#)
- [处理 XML 文件](#)
- [文档标记分隔符](#)
- [如何：使用 XML 文档功能](#)

相关章节

有关更多信息，请参见：

- [/doc\(处理文档注释\)](#)
- [“XML 文档”示例](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [附录 A: 文档注释](#)

请参见

概念

[C# 编程指南](#)

建议的文档注释标记(C# 编程指南)

C# 编译器将代码中的文档注释处理到 XML 文件中。需要在您的站点实现的具体内容是处理 XML 文件以创建文档。

在代码构造(如类型和类型成员)上处理标记。

注意

文档注释不能应用于命名空间。

编译器将处理任何为有效 XML 的标记。下列标记提供了用户文档中常用的功能：

<code><c></code>	<code><para></code>	<code><see>*</code>
<code><code></code>	<code><param>*</code>	<code><seealso>*</code>
<code><example></code>	<code><paramref></code>	<code><summary></code>
<code><exception>*</code>	<code><permission>*</code>	<code><typeparam>*</code>
<code><include>*</code>	<code><remarks></code>	<code><typeparamref></code>
<code><list></code>	<code><returns></code>	<code><value></code>

(* 表示编译器验证语法。)

如果希望尖括号显示在文档注释的文本中，请使用 < 和 >。例如，<尖括号中的文本>。

请参见

任务

["XML 文档"示例](#)

参考

[/doc\(处理文档注释\)\(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

[XML 文档注释\(C# 编程指南\)](#)

[XML 文档注释\(C# 编程指南\)](#)

<c>(C# 编程指南)

```
<c>text</c>
```

参数

text

希望将其指示为代码的文本。

备注

<c> 标记为您提供了一种将说明中的文本标记为代码的方法。使用 <code> 将多行指示为代码。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary><c>DoWork</c> is a method in the <c>TestClass</c> class.
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<code>(C# 编程指南)

```
<code>content</code>
```

参数

content

希望将其标记为代码的文本。

备注

<code> 标记为您提供了一种将多行指示为代码的方法。使用 <c> 指示应将说明中的文本标记为代码。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

有关如何使用 <code> 标记的示例, 请参见 [<example>](#) 主题。

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<example>(C# 编程指南)

```
<example>description</example>
```

参数

description

代码示例的说明。

备注

使用 <example> 标记可以指定使用方法或其他库成员的示例。这通常涉及使用 <code> 标记。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>
    /// The GetZero method.
    /// </summary>
    /// <example> This sample shows how to call the GetZero method.
    /// <code>
    /// class TestClass
    /// {
    ///     static int Main()
    ///     {
    ///         return GetZero();
    ///     }
    /// }
    /// </code>
    /// </example>
    public static int GetZero()
    {
        return 0;
    }

    /// text for Main
    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<exception> (C# 编程指南)

```
<exception cref="member">description</exception>
```

参数

`cref = "member"`

对可从当前编译环境中获取的异常的引用。编译器检查到给定异常存在后，将 `member` 转换为输出 XML 中的规范化元素名。必须将 `member` 括在双引号 (" ") 中。

有关如何创建对泛型类型的 `cref` 引用的更多信息，请参见 [<see> \(C# 编程指南\)](#)。

`description`

异常的说明。

备注

<exception> 标记使您可以指定哪些异常可被引发。此标记可用在方法、属性、事件和索引器的定义中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// comment for class
public class EClass : System.Exception
{
    // class definition...
}

/// comment for class
class TestClass
{
    /// <exception cref="System.Exception">Thrown when...</exception>
    public void DoSomething()
    {
        try
        {
        }
        catch (EClass)
        {
        }
    }
}
```

请参见

参考

[建议的文档注释标记 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<include>(C# 编程指南)

```
<include file='filename' path='tagpath[@name="id"]' />
```

参数

filename

包含文档的文件名。该文件名可用路径加以限定。将 *filename* 括在单引号 (' ') 中。

tagpath

filename 中指向标记 *name* 的标记路径。将此路径括在单引号中 (' ')。

name

注释前边的标记中的名称说明符;*name* 具有一个 *id*。

id

位于注释之前的标记的 ID。将此 ID 括在双引号中 (" ")。

备注

<include> 标记使您得以引用描述源代码中类型和成员的另一文件中的注释。这是除了将文档注释直接置于源代码文件中之外的另一种可选方法。

<include> 标记使用 XML XPath 语法。有关自定义 <include> 使用的方法, 请参见 XPath 文档。

示例

以下是一个多文件示例。第一个文件使用 <include>, 如下所列:

C#

```
// compile with: /doc:DocFileName.xml

/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    static void Main()
    {
    }
}

/// <include file='xml_include_tag.doc' path='MyDocs/MyMembers[@name="test2"]/*' />
class Test2
{
    public void Test()
    {
    }
}
```

第二个文件 xml_include_tag.doc 包含下列文档注释:

```
<MyDocs>

<MyMembers name="test">
<summary>
The summary for this type.
</summary>
</MyMembers>

<MyMembers name="test2">
<summary>
The summary for this other type.
</summary>
</MyMembers>
```

```
</summary>
</MyMembers>

</MyDocs>
```

程序输出

```
<?xml version="1.0"?>
<doc>
<assembly>
<name>xml_include_tag</name>
</assembly>
<members>
<member name="T:Test">
<summary>
The summary for this type.
</summary>
</member>
<member name="T:Test2">
<summary>
The summary for this other type.
</summary>
</member>
</members>
</doc>
```

请参见

参考

[建议的文档注释标记 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<list>(C# 编程指南)

```
<list type="bullet" | "number" | "table">
  <listheader>
    <term>term</term>
    <description>description</description>
  </listheader>
  <item>
    <term>term</term>
    <description>description</description>
  </item>
</list>
```

参数

term

要定义的项，该项将在 *description* 中定义。

description

项目符号列表或编号列表中的项或者 *term* 的定义。

备注

<listheader> 块用于定义表或定义列表中的标题行。定义表时，只需为标题中的项提供一个项。

列表中的每一项都用一个 <item> 块来描述。创建定义列表时，既需要指定 *term* 也需要指定 *description*。但是，对于表、项目符号列表或编号列表，只需为 *description* 提供一个项。

列表或表所拥有的 <item> 块数可以根据需要而定。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
  /// <summary>Here is an example of a bulleted list:
  /// <list type="bullet">
  ///   <item>
  ///     <description>Item 1.</description>
  ///   </item>
  ///   <item>
  ///     <description>Item 2.</description>
  ///   </item>
  /// </list>
  /// </summary>
  static void Main()
  {
  }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<para>(C# 编程指南)

```
<para>content</para>
```

参数

content

段落文本。

备注

<para> 标记用于诸如 <summary>、<remarks> 或 <returns> 等标记内，使您得以将结构添加到文本中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

有关使用 <para> 的示例，请参见 [<summary>](#)。

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<param>(C# 编程指南)

```
<param name='name'>description</param>
```

参数

name

方法参数名。将此名称用双引号括起来 (" ")。

description

参数说明。

备注

<param> 标记应当用于方法声明的注释中，以描述方法的一个参数。

有关 <param> 标记的文本将显示在 [IntelliSense](#)、对象浏览器和代码注释 Web 报表中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <param name="Int1">Used to indicate status.</param>
    public static void DoWork(int Int1)
    {
    }
    /// text for Main
    static void Main()
    {
    }
}
```

[请参见](#)

[参考](#)

[建议的文档注释标记\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

<paramref> (C# 编程指南)

```
<paramref name="name"/>
```

参数

name

要引用的参数名。将此名称用双引号括起来 (" ")。

备注

<paramref> 标记提供了指示代码注释中的某个单词(例如在 <summary> 或 <remarks> 块中)引用某个参数的方式。可以处理 XML 文件来以不同的方式格式化此单词, 比如将其设置为粗体或斜体。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>DoWork is a method in the TestClass class.
    /// The <paramref name="Int1"/> parameter takes a number.
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<permission>(C# 编程指南)

```
<permission cref="member">description</permission>
```

参数

cref = "member"

对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 *member* 转换为输出 XML 中的规范化元素名。必须将 *member* 括在双引号 (" ") 中。

有关如何创建对泛型类型的 cref 引用的信息，请参见 [<see> \(C# 编程指南\)](#)。

description

对成员的访问的说明。

备注

<permission> 标记使您得以将成员的访问记入文档。使用 [PermissionSet](#) 类可以指定对成员的访问。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

class TestClass
{
    /// <permission cref="System.Security.PermissionSet">Everyone can access this method.</permission>
    public static void Test()
    {
    }

    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<remarks>(C# 编程指南)

```
<remarks>description</remarks>
```

参数

Description

成员的说明。

备注

<remarks> 标记用于添加有关某个类型的信息，从而补充由 <summary> 所指定的信息。此信息显示在[对象浏览器](#)中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// <summary>
/// You may have some primary information about this class.
/// </summary>
/// <remarks>
/// You may have some additional information about this class.
/// </remarks>
public class TestClass
{
    /// text for Main
    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<returns>(C# 编程指南)

```
<returns>description</returns>
```

参数

description

返回值的说明。

备注

<returns> 标记应当用于方法声明的注释，以描述返回值。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <returns>Returns zero.</returns>
    public static int GetZero()
    {
        return 0;
    }

    /// text for Main
    static void Main()
    {
    }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<see>(C# 编程指南)

```
<see cref="member"/>
```

参数

cref = "member"

对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查给定的代码元素是否存在，并将 *member* 传递给输出 XML 中的元素名称。应将 *member* 放在双引号 (" ") 中。

备注

<see> 标记使您得以从文本内指定链接。使用 <seealso> 指示文本应该放在“另请参见”节中。

使用 /doc 进行编译可以将文档注释处理到文件中。

有关使用 <see> 的示例，请参见 <summary>。

示例

下面的示例演示如何对泛型类型进行 cref 引用。

C#

```
// compile with: /doc:DocFileName.xml

// the following cref shows how to specify the reference, such that,
// the compiler will resolve the reference
/// <summary cref="C{T}">
/// </summary>
class A { }

// the following cref shows another way to specify the reference,
// such that, the compiler will resolve the reference
// <summary cref="C &lt; T &gt;">

// the following cref shows how to hard-code the reference
/// <summary cref="T:C`1">
/// </summary>
class B { }

/// <summary cref="A">
/// </summary>
/// <typeparam name="T"></typeparam>
class C<T> { }
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<seealso>(C# 编程指南)

```
<seealso cref="member"/>
```

参数

cref = "member"

对可以通过当前编译环境进行调用的成员或字段的引用。编译器检查到给定代码元素存在后，将 *member* 传递给输出 XML 中的元素名。必须将 *member* 括在双引号 (" ") 中。

有关如何创建对泛型类型的 cref 引用的信息，请参见 [<see> \(C# 编程指南\)](#)。

备注

<seealso> 标记使您得以指定希望在“请参见”一节中出现的文本。使用 [<see>](#) 从文本内指定链接。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

有关使用 <seealso> 的示例，请参见 [<summary>](#)。

[请参见](#)

[参考](#)

[建议的文档注释标记 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

<summary>(C# 编程指南)

```
<summary>description</summary>
```

参数

description

对象的摘要。

备注

<summary> 标记应当用于描述类型或类型成员。使用 <remarks> 添加针对某个类型说明的补充信息。

<summary> 标记的文本是唯一有关 IntelliSense 中的类型的信息源，它也显示在[对象浏览器](#)中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class TestClass
public class TestClass
{
    /// <summary>DoWork is a method in the TestClass class.
    /// <para>Here's how you could make a second paragraph in a description. <see cref="System.Console.WriteLine(System.String)"/> for information about output statements.</para>
    /// <seealso cref="TestClass.Main"/>
    /// </summary>
    public static void DoWork(int Int1)
    {
    }

    /// text for Main
    static void Main()
    {
    }
}
```

[请参见](#)

[参考](#)

[建议的文档注释标记\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

<typeparam>(C# 编程指南)

```
<typeparam name="name">description</typeparam>
```

参数

name

类型参数的名称。将此名称用双引号括起来 (" ")。

description

类型参数的说明。

备注

在泛型类型或方法声明的注释中应该使用 <**typeparam**> 标记描述类型参数。为泛型类型或方法的每个类型参数添加标记。

有关更多信息，请参见[泛型\(C# 编程指南\)](#)。

<**typeparam**> 标记的文本将显示在[对象浏览器](#)代码注释 Web 报表 IntelliSense 中。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

有关显示如何使用 <**typeparam**> 的代码示例，请参见[<typeparamref>\(C# 编程指南\)](#)。

[请参见](#)

[参考](#)

[建议的文档注释标记\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

<typeparamref> (C# 编程指南)

```
<typeparamref name="name"/>
```

参数

name

类型参数的名称。将此名称用双引号括起来 (" ")。

备注

有关泛型类型和方法中的类型参数的更多信息，请参见[泛型](#)。

使用此标记，文档文件的使用者能够以某种独特的方式设置单词的格式，例如以斜体显示。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// comment for class
public class TestClass
{
    /// <summary>
    /// Creates a new array of arbitrary type <typeparamref name="T"/>
    /// </summary>
    /// <typeparam name="T">The element type of the array</typeparam>
    public static T[] mkArray<T>(int n)
    {
        return new T[n];
    }
}
```

请参见

参考

[建议的文档注释标记 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

<value>(C# 编程指南)

```
<value>property-description</value>
```

参数

property-description

属性的说明。

备注

<value> 标记使您得以描述属性所代表的值。请注意，当在 Visual Studio .NET 开发环境中通过代码向导添加属性时，它将会为新属性添加 <summary> 标记。然后，应该手动添加 <value> 标记以描述该属性所表示的值。

使用 [/doc](#) 进行编译可以将文档注释处理到文件中。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// text for class Employee
public class Employee
{
    private string _name;

    /// <summary>The Name property represents the employee's name.</summary>
    /// <value>The Name property gets/sets the _name data member.</value>
    public string Name
    {
        get
        {
            return _name;
        }
        set
        {
            _name = value;
        }
    }

    /// text for class MainClass
    public class MainClass
    {
        /// text for Main
        static void Main()
        {
        }
    }
}
```

请参见

参考

[建议的文档注释标记\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

处理 XML 文件 (C# 编程指南)

编译器为代码中被标记为生成文档的每一个构造生成一个 ID 字符串。(有关如何标记代码的信息, 请参见[建议的文档注释标记](#)。)ID 字符串唯一地标识构造。处理 XML 文件的程序可以使用 ID 字符串标识文档应用于的相应 .NET Framework 元数据/反射项。

XML 文件不是代码的分层表示形式, 它是一个平面列表, 每一个元素都有一个生成的 ID。

编译器在生成 ID 字符串时遵循下列规则:

- 不在字符串中放置空白。
- ID 字符串的第一部分通过单个字符后跟一个冒号来标识被标识成员的种类。使用下列成员类型:

字符	说明
N	命名空间 不可将文档注释添加到命名空间中, 但是可以对它们进行 cref 引用(在受支持的位置)。
T	类型:类、接口、结构、枚举、委托
F	字段
P	属性(包括索引程序或其他索引属性)
M	方法(包括一些特殊方法, 例如构造函数、运算符等)
E	事件
!	错误字符串 字符串的其余部分会提供有关此错误的信息。C# 编译器为无法解析的链接生成错误信息。

- 字符串的第二部分是项的完全限定名, 从命名空间的根开始。项的名称、其封闭类型和命名空间以句号分隔。如果项的名称本身包含句号, 则用哈希符号 (#) 替换这些句号。假定任何项的名称中都不直接存在哈希符号。例如, String 构造函数的完全限定名是 "System.String.#ctor"。
- 对于属性和方法, 如果该方法带参数, 则将其后的参数列表括在括号中。如果没有参数, 则没有括号。多个参数以逗号分隔。每个参数的编码都直接遵循它在 .NET Framework 签名中的编码方法:
 - 基类型。常规类型(ELEMENT_TYPE_CLASS 或 ELEMENT_TYPE_VALUETYPE)表示为类型的完全限定名。
 - 内部类型(例如 ELEMENT_TYPE_I4、ELEMENT_TYPE_OBJECT、ELEMENT_TYPE_STRING、ELEMENT_TYPE_TYPEDBYREF 和 ELEMENT_TYPE_VOID)表示为相应完全类型的完全限定名。例如, System.Int32 或 System.TypedReference。
 - ELEMENT_TYPE_PTR 表示为 "*", 在修改的类型之后。
 - ELEMENT_TYPE_BYREF 表示为 "@", 在修改的类型之后。
 - ELEMENT_TYPE_PINNED 表示为 "^", 在修改的类型之后。C# 编译器永远不会生成此结果。
 - ELEMENT_TYPE_CMOD_REQ 表示为 "|" 和修饰符类的完全限定名, 在修改的类型之后。C# 编译器永远不会生成此结果。
 - ELEMENT_TYPE_CMOD_OPT 表示为 "!" 和修饰符类的完全限定名, 在修改的类型之后。
 - ELEMENT_TYPE_SZARRAY 表示为 "[]", 在数组的元素类型之后。
 - ELEMENT_TYPE_GENERICARRAY 表示为 "[?]", 在数组的元素类型之后。C# 编译器永远不会生成此结果。
 - ELEMENT_TYPE_ARRAY 表示为 [lowerbound:size,lowerbound:size], 其中逗号数为秩 - 1, 每一维的下限和大小(如果已知)用十进制表示。如果未指定下限及大小, 它将完全被省略。如果省略了特定维的下限及大小, 则 ":" 也将被

省略。例如，以 1 作为下限并且未指定大小的二维数组是 [1;1:]。

- ELEMENT_TYPE_FNPTR 表示为“=FUNC:type(signature)”，其中 type 是返回类型，signature 是方法的参数。如果没有参数，则将省略括号。C# 编译器永远不会生成此结果。

不表示下列签名组件，因为从不使用它们来区分重载方法：

- 调用约定
 - 返回类型
 - ELEMENT_TYPE_SENTINEL
- 仅对于转换运算符(op_Implicit 和 op_Explicit)，方法的返回值才被编码为“~”，后跟返回类型，如上述编码所示。
 - 对于泛型类型，类型名称后跟反勾号，再跟一个数字，指示泛型类型参数的个数。例如，

<member name="T:SampleClass`2"> 是定义为 public class SampleClass<T, U> 的类型的标记。

对于接受泛型类型作为参数的方法，泛型类型参数被指定为数字前面加上反勾号(如 `0、`1)。表示类型泛型参数的数组表示法(从零开始)的每个数字。

示例

下列示例显示了如何生成类及其成员的 ID 字符串：

C#

```
///  
///  
namespace N // "N:N"  
{  
    ///  
    ///  
    public unsafe class X // "T:N.X"  
    {  
        public X(){}
        //-----  
        // The result of the above is:  
        // "M:N.X.#ctor"  
  
        /// <param name="i"></param>  
        public X(int i){}
        //-----  
        // The result of the above is:  
        // "M:N.X.#ctor(System.Int32)"  
  
        ~X(){}
        //-----  
        // The result of the above is:  
        // "M:N.X.Finalize", destructor's representation in metadata  
  
        public string q;
        //-----  
        // The result of the above is:  
        // "F:N.X.q"  
  
        /// <returns></returns>
        public const double PI = 3.14;
        //-----  
        // The result of the above is:  
        // "F:N.X.PI"  
  
        /// <param name="s"></param>
        /// <param name="y"></param>
        /// <param name="z"></param>
```

```
/// <returns></returns>
public int f(){return 1;}
//-----
// The result of the above is:
// "M:N.X.f"

/// <param name="array1"></param>
/// <param name="array"></param>
/// <returns></returns>
public int bb(string s, ref int y, void * z){return 1;}
//-----
// The result of the above is:
// "M:N.X.bb(System.String,System.Int32@,=System.Void*)"

/// <param name="x"></param>
/// <param name="xx"></param>
/// <returns></returns>
public int gg(short[] array1, int[,] array){return 0;}
//-----
// The result of the above is:
// "M:N.X.gg(System.Int16[], System.Int32[0:,0:])"

public static X operator+(X x, X xx){return x;}
//-----
// The result of the above is:
// "M:N.X.op_Addition(N.X,N.X)"

public int prop {get{return 1;} set{}}
//-----
// The result of the above is:
// "P:N.X.prop"

public event D d;
//-----
// The result of the above is:
// "E:N.X.d"

public int this[string s]{get{return 1;}}
//-----
// The result of the above is:
// "P:N.X.Item(System.String)"

public class Nested{}
//-----
// The result of the above is:
// "T:N.X.Nested"

public delegate void D(int i);
//-----
// The result of the above is:
// "T:N.X.D"

/// <param name="x"></param>
/// <returns></returns>
public static explicit operator int(X x){return 1;}
//-----
// The result of the above is:
// "M:N.X.op_Explicit(N.X)~System.Int32"
}
```

}

请参见

任务

["XML 文档"示例](#)

参考

[/doc\(处理文档注释\)\(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

[XML 文档注释\(C# 编程指南\)](#)

文档标记的分隔符(C# 编程指南)

使用 XML doc 注释需要分隔符，分隔符为编译器指示文档注释的起止位置。XML 文档标记可以与以下几种分隔符一起使用：

///

这是显示在文档示例中并由 Visual C# 项目模板使用的形式。

注意

Visual Studio IDE 具有一项称为“智能注释编辑”的功能，它自动插入 `<summary>` 和 `</summary>` 标记，并且当您在代码编辑器中键入 `///` 分隔符后将光标移动到这些标记内。从项目属性页的“选项”对话框 -> “文本编辑器”->“C#/J#”->“格式设置”可以访问此功能。

`/** * /`

多行分隔符。

使用 `/** * /` 分隔符时有一些格式设置规则：

- 对于包含 `/**` 分隔符的行，如果该行的其余部分为空白，则不将该行处理为注释。如果第一个字符为空白，则忽略此空白字符并处理该行的剩余部分。否则，将 `/**` 分隔符后的整行文本处理为注释的一部分。
- 对于包含 `*/` 分隔符的行，如果到 `*/` 分隔符为止仅有空白，则忽略该行。否则，将到 `*/` 分隔符为止的那行文本处理为注释的一部分，同时必须遵循以下描述的模式匹配规则。
- 对于以 `/**` 分隔符开头的行后面的那些行，编译器会在每个行的开头寻找一个公共模式，这些行由可选的空白和星号 (*) 以及后面更多可选的空白组成。如果编译器在每行的开头均找到了一组公共的字符，则它将对 `/**` 分隔符到包含 `*/` 分隔符的行(可能包括此行)之间的所有行忽略该模式。

示例：

- 以下注释中将被处理的唯一部分是以 `<summary>` 开头的行。下列两个标记格式将产生相同的注释：

```
/**  
 <summary>text</summary>  
 */  
/** <summary>text</summary> */
```

- 编译器在第二行和第三行的开头应用“*”模式进行忽略。

```
/**  
 * <summary>  
 * text </summary>*/
```

- 编译器在此注释中找不到任何模式，因为第二行没有星号。因而，第二行和第三行直到 `*/` 的所有文本都将被处理为注释的一部分。

```
/**  
 * <summary>  
 text </summary>*/
```

- 编译器之所以在此注释中没找到任何模式有两个原因。首先，行首星号前的空格数目不一致。其次，第五行是以 Tab 开头的，它并不等效于空格。因而，从第二行到 `*/` 的所有文本都将被处理为注释的一部分。

```
/**  
 * <summary>  
 * text  
 * text2
```

* </summary>

*/

请参见

任务

["XML 文档"示例](#)

参考

[/doc\(处理文档注释\)\(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

[XML 文档注释\(C# 编程指南\)](#)

[XML 文档注释\(C# 编程指南\)](#)

如何: 使用 XML 文档功能 (C# 编程指南)

下面的示例提供对某个已存档的类型的基本概述。

示例

C#

```
// compile with: /doc:DocFileName.xml

/// <summary>
/// Class level summary documentation goes here.</summary>
/// <remarks>
/// Longer comments can be associated with a type or member
/// through the remarks tag</remarks>
public class TestClass
{
    /// <summary>
    /// Store for the name property</summary>
    private string _name = null;

    /// <summary>
    /// The class constructor. </summary>
    public TestClass()
    {
        // TODO: Add Constructor Logic here
    }

    /// <summary>
    /// Name property </summary>
    /// <value>
    /// A value tag is used to describe the property value</value>
    public string Name
    {
        get
        {
            if (_name == null)
            {
                throw new System.Exception("Name is null");
            }
            return _name;
        }
    }

    /// <summary>
    /// Description for SomeMethod.</summary>
    /// <param name="s"> Parameter description for s goes here</param>
    /// <seealso cref="System.String">
    /// You can use the cref attribute on any tag to reference a type or member
    /// and the compiler will check that the reference exists. </seealso>
    public void SomeMethod(string s)
    {
    }

    /// <summary>
    /// Some other method. </summary>
    /// <returns>
    /// Return results are described through the returns tag.</returns>
    /// <seealso cref="SomeMethod(string)">
    /// Notice the use of the cref attribute to reference a specific method </seealso>
    public int SomeOtherMethod()
    {
        return 0;
    }

    /// <summary>
    /// The entry point for the application.
    /// </summary>
}
```

```

    /// </summary>
    /// <param name="args"> A list of command line arguments</param>
    static int Main(System.String[] args)
    {
        // TODO: Add code to start application here
        return 0;
    }
}

```

示例输出

```

// This .xml file was generated with the previous code sample.
<?xml version="1.0"?>
<doc>
    <assembly>
        <name>xmlsample</name>
    </assembly>
    <members>
        <member name="T:SomeClass">
            <summary>
                Class level summary documentation goes here.</summary>
            <remarks>
                Longer comments can be associated with a type or member
                through the remarks tag</remarks>
            </member>
        <member name="F:SomeClass.m_Name">
            <summary>
                Store for the name property</summary>
            </member>
        <member name="M:SomeClass.#ctor">
            <summary>The class constructor.</summary>
        </member>
        <member name="M:SomeClass.SomeMethod(System.String)">
            <summary>
                Description for SomeMethod.</summary>
            <param name="s"> Parameter description for s goes here</param>
            <seealso cref="T:System.String">
                You can use the cref attribute on any tag to reference a type or member
                and the compiler will check that the reference exists. </seealso>
            </member>
        <member name="M:SomeClass.SomeOtherMethod">
            <summary>
                Some other method. </summary>
            <returns>
                Return results are described through the returns tag.</returns>
            <seealso cref="M:SomeClass.SomeMethod(System.String)">
                Notice the use of the cref attribute to reference a specific method </seealso>
            </member>
        <member name="M:SomeClass.Main(System.String[])">
            <summary>
                The entry point for the application.
            </summary>
            <param name="args"> A list of command line arguments</param>
        </member>
        <member name="P:SomeClass.Name">
            <summary>
                Name property </summary>
            <value>
                A value tag is used to describe the property value</value>
            </member>
    </members>
</doc>

```

编译代码

若要编译该示例，请键入以下命令行：

```
csc XMLsample.cs /doc:XMLsample.xml
```

这将创建 XML 文件 XMLsample.xml，您可以在浏览器中或使用 TYPE 命令查看该文件。

可靠编程

XML 文档以 `///` 开头。创建新项目时，向导将为您放入一些起始 `///` 行。对这些注释的处理有一些限制：

- 文档必须是格式良好的 XML。如果 XML 格式不良，将生成警告，并且文档文件将包含一条注释，指出遇到错误。
- 开发人员可自由创建自己的标记集。有一套建议的标记（请参见“其他阅读材料”部分）。某些建议的标记具有特殊含义：
 - `<param>` 标记用于描述参数。如果使用，编译器将验证参数是否存在，以及文档中是否描述了所有参数。如果验证失败，则编译器发出警告。
 - `cref` 属性可以附加到任意标记，以提供对代码元素的引用。编译器将验证该代码元素是否存在。如果验证失败，则编译器发出警告。查找 `cref` 属性中描述的类型时，编译器考虑所有 `using` 语句。
 - `<summary>` 标记由 Visual Studio 内的“Intellisense”使用，用来显示类型或成员的其他相关信息。

注意

XML 文件并不提供有关类型和成员的完整信息（例如，它不包含任何类型信息）。若要获得有关类型或成员的完整信息，文档文件必须与实际类型或成员上的反射一起使用。

请参见

任务

[“XML 文档”示例](#)

参考

[/doc\(处理文档注释\)\(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

[XML 文档注释\(C# 编程指南\)](#)

[XML 文档注释\(C# 编程指南\)](#)

应用程序域 (C# 编程指南)

应用程序域为隔离正在运行的应用程序提供了一种灵活而安全的方法。

应用程序域通常由运行库宿主创建和操作。有时，您可能希望应用程序以编程方式与应用程序域交互，例如想在不停止应用程序运行的情况下卸载某个组件时。

应用程序域使应用程序以及应用程序的数据彼此分离，有助于提高安全性。单个进程可以运行多个应用程序域，并具有在单独进程中所存在的隔离级别。在单个进程中运行多个应用程序提高了服务器伸缩性。

下面的代码示例创建一个新的应用程序域，然后加载并执行以前生成的程序集 `HelloWorld.exe`，该程序集存储在驱动器 C 上。

C#

```
static void Main()
{
    // Create an Application Domain:
    System.AppDomain newDomain = System.AppDomain.CreateDomain("NewApplicationDomain");

    // Load and execute an assembly:
    newDomain.ExecuteAssembly(@"c:\HelloWorld.exe");

    // Unload the application domain:
    System.AppDomain.Unload(newDomain);
}
```

应用程序域概述

应用程序域具有以下特点：

- 必须先将程序集加载到应用程序域中，然后才能执行该程序集。有关更多信息，请参见[程序集和全局程序集缓存 \(C# 编程指南\)](#)。
- 一个应用程序域中的错误不会影响在另一个应用程序域中运行的其他代码。
- 能够在不停止整个进程的情况下停止单个应用程序并卸载代码。不能卸载单独的程序集或类型，只能卸载整个应用程序域。

相关章节

- [应用程序域概述](#)
- [应用程序域](#)
- [应用程序域和程序集](#)
- [对应用程序域进行编程](#)
- [使用应用程序域和程序集编程](#)
- [在另一个应用程序域中执行代码 \(C# 编程指南\)](#)
- [如何：创建和使用应用程序域 \(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 3.1 应用程序启动

请参见

概念

[C# 编程指南](#)

[程序集和全局程序集缓存 \(C# 编程指南\)](#)

在另一个应用程序域中执行代码 (C# 编程指南)

一旦将程序集加载到应用程序域中，就可以执行该程序集所包含的代码。最简单的加载方法是使用 [AssemblyLoad](#)，它会将程序集加载到当前应用程序域中，并从程序集的默认入口点开始运行代码。

如果希望将该程序集加载到另外一个应用程序域中，可以使用 [ExecuteAssembly](#) 或 [ExecuteAssemblyByName](#)，或者使用这些方法的其他重载版本之一。

如果想从默认入口点以外的位置执行另一个程序集，可在远程程序集中定义一个从 [MarshalByRefObject](#) 派生的新类型。然后在应用程序中使用 [CreateInstance](#) 创建该类型的一个实例。

请考虑下面的文件，它创建一个包含一个命名空间和两个类的程序集。假设此程序集已经生成，并以 `HelloWorldRemote.exe` 为名存储在驱动器 C 上。

C#

```
// This namespace contains code to be called.
namespace HelloWorldRemote
{
    public class RemoteObject : System.MarshalByRefObject
    {
        public RemoteObject()
        {
            System.Console.WriteLine("Hello, World! (RemoteObject Constructor)");
        }
        class Program
        {
            static void Main()
            {
                System.Console.WriteLine("Hello, World! (Main method)");
            }
        }
    }
}
```

为了从其他应用程序访问该代码，可以将该程序集加载到当前应用程序域中，或创建新的应用程序域并将该程序集加载到其中。如果使用 [Assembly.LoadFrom](#) 将程序集加载到当前应用程序域中，您可以使用 [Assembly.CreateInstance](#) 来实例化 [RemoteObject](#) 类的实例，这样将导致执行对象构造函数。

C#

```
static void Main()
{
    // Load the assembly into the current appdomain:
    System.Reflection.Assembly newAssembly = System.Reflection.Assembly.LoadFrom(@"c:\HelloWorldRemote.exe");

    // Instantiate RemoteObject:
    newAssembly.CreateInstance("HelloWorldRemote.RemoteObject");
}
```

将程序集加载到一个单独的应用程序域时，应使用 [AppDomain.ExecuteAssembly](#) 来访问默认入口点，或使用 [AppDomain.CreateInstance](#) 创建 [RemoteObject](#) 类的实例。创建该实例将导致执行构造函数。

C#

```
static void Main()
{
    System.AppDomain NewAppDomain = System.AppDomain.CreateDomain("NewApplicationDomain");

    // Load the assembly and call the default entry point:
    NewAppDomain.ExecuteAssembly(@"c:\HelloWorldRemote.exe");

    // Create an instance of RemoteObject:
}
```

```
        NewAppDomain.CreateInstanceFrom(@"c:\HelloWorldRemote.exe", "HelloWorldRemote.RemoteObj
ect");
    }
```

如果不想以编程方式加载程序集，可以从“解决方案资源管理器”中使用“添加引用”来指定程序集 HelloWorldRemote.exe。然后向应用程序的 `using` 块中添加一个 `using HelloWorldRemote;` 指令，并在程序中使用 `RemoteObject` 类型来声明 `RemoteObject` 对象的一个实例，如下所示：

C#

```
static void Main()
{
    // This code creates an instance of RemoteObject, assuming HelloWorldRemote has been ad
ded as a reference:
    HelloWorldRemote.RemoteObject o = new HelloWorldRemote.RemoteObject();
}
```

请参见

参考

[应用程序域 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[应用程序域概述](#)

[应用程序域和程序集](#)

[对应用程序域进行编程](#)

其他资源

[应用程序域](#)

[使用应用程序域和程序集编程](#)

如何: 创建和使用应用程序域 (C# 编程指南)

出于安全和性能目的, 应用程序域提供了隔离代码的方法。有关更多信息, 请参见[应用程序域 \(C# 编程指南\)](#)。

使用应用程序域

1. 创建应用程序域。公共语言运行库宿主自动为应用程序创建一个应用程序域。有关信息, 请参见主题[如何: 创建应用程序域](#)。
2. 配置应用程序域。有关更多信息, 请参见[如何: 配置应用程序域](#)。
3. 将程序集加载到应用程序域中。有关更多信息, 请参见[如何: 将程序集加载到应用程序域中](#)。
4. 访问其他程序集的内容。有关更多信息, 请参见[在另一个应用程序域中执行代码 \(C# 编程指南\)](#)。
5. 卸载应用程序域。有关更多信息, 请参见[如何: 卸载应用程序域](#)。

请参见

参考

[应用程序域 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[应用程序域概述](#)

[应用程序域和程序集](#)

[对应用程序域进行编程](#)

其他资源

[应用程序域](#)

[使用应用程序域和程序集编程](#)

程序集和全局程序集缓存 (C# 编程指南)

程序集是任何 .NET Framework 应用程序的基本构造块。例如，在生成简单的 C# 应用程序时，Visual Studio 创建一个单个可移植可执行 (PE) 文件形式的程序集，明确地说就是一个 EXE 或 DLL。

程序集包含描述它们自己的内部版本号和它们包含的所有数据和对象类型的详细信息的元数据。有关更多信息，请参见[程序集清单](#)。

程序集仅在需要时才加载。如果不使用程序集，则不会加载。这意味着程序集可能是在大型项目中管理资源的有效途径。

程序集可以包含一个或多个模块。例如，计划较大的项目时，可以让几个各个开发人员负责单独的模块，并通过组合所有这些模块来创建单个程序集。有关模块的更多信息，请参见主题[如何：生成多文件程序集](#)。

程序集概述

程序集具有以下特点：

- 程序集作为 .exe 或 .dll 文件实现。
- 通过将程序集放在全局程序集缓存中，可在多个应用程序之间共享程序集。
- 要将程序集放在全局程序集缓存中，必须对程序集进行强命名。有关更多信息，请参见[具有强名称的程序集](#)。
- 程序集仅在需要时才加载到内存中。
- 可以使用反射来以编程方式获取关于程序集的信息。有关更多信息，请参见主题[反射](#)。
- 如果加载程序集的目的只是对其进行检查，应使用诸如 `ReflectionOnlyLoadFrom` 的方法。
- 可以在单个应用程序中使用相同程序集的两个版本。有关更多信息，请参见 `extern` 别名。

相关章节

更多信息：

- [友元程序集 \(C# 编程指南\)](#)
- [如何：与其他应用程序共享程序集 \(C# 编程指南\)](#)
- [如何：加载和卸载程序集 \(C# 编程指南\)](#)
- [如何：确定文件是否为程序集 \(C# 编程指南\)](#)
- [extern \(C# 参考\)](#)
- [公共语言运行库中的程序集](#)
- [程序集概述](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.2 程序结构
- 9.1 编译单元

请参见

参考

[应用程序域 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[元数据和 PE 文件结构](#)

友元程序集 (C# 编程指南)

可以从一个程序集访问另一个程序集中的内部类型或内部成员。

备注

友元程序集功能用于访问内部成员;私有类型和私有成员仍然不可访问。

若要使程序集(程序集 B)能够访问另一个程序集(程序集 A)的内部类型和成员, 应使用程序集 A 中的 [InternalsVisibleToAttribute 属性](#)。

注意

在对将要访问另一个程序集(程序集 A)的内部类型或内部成员的程序集(程序集 B)进行编译时, 必须用 [/out](#) 编译器选项显式指定输出文件的名称(.exe 或 .dll)(有关更多信息, 请参见 [/out](#))。这是必需的, 因为当编译器将生成的程序集绑定到外部引用时, 尚未为该程序集生成名称。

[StrongNameIdentityPermission](#) 类还提供共享类型的功能, 其与友元程序集的区别如下:

- **StrongNameIdentityPermission** 应用于单个类型, 而友元程序集应用于整个程序集。
- 如果希望程序集 B 能够共享程序集 A 中的数百个类型, 则必须使用 **StrongNameIdentityPermission** 修饰所有这些类型;而使用友元程序集时只需声明一次友元关系。
- 使用 **StrongNameIdentityPermission** 时, 想要共享的类型必须声明为公共类型。使用友元程序集时, 会将共享类型声明为内部类型。
- 有关如何生成可访问程序集中的非公共类型的 .netmodule 的信息, 请参见 [/moduleassemblyname](#)。

示例

在此示例中, 程序集使内部类型和内部成员对名为 called cs_friend_assemblies_2 的程序集可见。

```
// cs_friend_assemblies.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
using System;

[assembly:InternalsVisibleTo("cs_friend_assemblies_2")]

// internal by default
class Class1
{
    public void Test()
    {
        Console.WriteLine("Class1.Test");
    }
}

// public type with internal member
public class Class2
{
    internal void Test()
    {
        Console.WriteLine("Class2.Test");
    }
}
```

在此示例中, 程序集使用程序集 cs_friend_assemblies.dll 中的内部类型和内部成员。

注意, 必须显式指定输出文件的名称 ([/out:cs_friend_assemblies_2.exe](#))。

如果此程序集允许另一个程序集(程序集 C)访问它的内部类型和成员, 则程序集 C 不会自动变成程序集 cs_friend_assemblies.dll 的友元。

```

// cs_friend_assemblies_2.cs
// compile with: /reference:cs_friend_assemblies.dll /out:cs_friend_assemblies_2.exe
public class M
{
    static void Main()
    {
        // access an internal type
        Class1 a = new Class1();
        a.Test();

        Class2 b = new Class2();
        // access an internal member of a public type
        b.Test();
    }
}

```

输出

```

Class1.Test
Class2.Test

```

此示例演示了如何使内部类型和成员对具有强名称的程序集可用。

若要生成 keyfile 并显示公钥, 请使用下面的 sn.exe 命令序列(有关更多信息, 请参见[强名称工具 \(Sn.exe\)](#)) :

- sn -k friend_assemblies.snk // 生成强名称密钥
- sn -p friend_assemblies.snk key.publickey // 将公钥从 key.snk 提取到 key.publickey 中
- sn -tp key.publickey // 显示存储在文件 key.publickey 中的公钥

用 [/keyfile](#) 将 keyfile 传递到编译器。

```

// cs_friend_assemblies_3.cs
// compile with: /target:library /keyfile:friend_assemblies.snk
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo("cs_friend_assemblies_4, PublicKey=002400000480000094000000060
200000240000525341310004000001000100031d7b6f3abc16c7de526fd67ec2926fe68ed2f9901afbc5f1b6b4
28bf6cd9086021a0b38b76bc340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09
172f7b62dabdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cbff2c58
1fc0f2d966b77114b2c4")]
class Class1
{
    public void Test()
    {
        System.Console.WriteLine("Class1.Test");
    }
}

```

此示例演示如何使用对具有强名称的程序集可用的内部类型和成员。

```

// cs_friend_assemblies_4.cs
// compile with: /keyfile:friend_assemblies.snk /reference:cs_friend_assemblies_3.dll /out:
cs_friend_assemblies_4.exe
public class M
{
    static void Main()
    {
        Class1 a = new Class1();
        a.Test();
    }
}

```

输出

Class1.Test

请参见

概念

[C# 编程指南](#)

[程序集和全局程序集缓存\(C# 编程指南\)](#)

如何：确定文件是否为程序集 (C# 编程指南)

当且仅当一个文件是托管文件并且在其元数据中包含程序集入口时，该文件才是一个程序集。有关程序集和元数据的更多信息，请参见主题[程序集清单](#)。

如何手动确定一个文件是否为程序集

1. 启动 [MSIL 反汇编程序 \(Ildasm.exe\)](#)。
2. 加载希望测试的文件。
3. 如果 ILDASM 报告该文件不是可移植的可执行 (PE) 文件，则它不是程序集。有关更多信息，请参见主题[如何：查看程序集内容](#)。

如何以编程方式确定一个文件是否为程序集

1. 调用 [GetAssemblyName](#) 方法，并向其传递正在测试的文件的完整文件路径和名称。
2. 如果引发 [BadImageFormatException](#) 异常，则该文件不是程序集。

示例

此示例测试一个 DLL 以确定它是否为程序集。

C#

```
class TestAssembly
{
    static void Main()
    {
        try
        {
            System.Reflection.AssemblyName testAssembly =
                System.Reflection.AssemblyName.GetAssemblyName(@"C:\WINDOWS\system\avicap.d
11");
            System.Console.WriteLine("Yes, the file is an Assembly.");
        }

        catch (System.IO.FileNotFoundException e)
        {
            System.Console.WriteLine("The file cannot be found.");
        }

        catch (System.BadImageFormatException e)
        {
            System.Console.WriteLine("The file is not an Assembly.");
        }

        catch (System.IO.FileLoadException e)
        {
            System.Console.WriteLine("The Assembly has already been loaded.");
        }
    }
}
```

[GetAssemblyName](#) 方法加载测试文件，然后在读取信息之后释放它。

输出

The file is not an Assembly.

请参见

任务

[关于异常的疑难解答 : System.BadImageFormatException](#)

[参考](#)

[AssemblyName](#)

[概念](#)

[C# 编程指南](#)

[程序集和全局程序集缓存 \(C# 编程指南\)](#)

如何：加载和卸载程序集 (C# 编程指南)

程序引用的程序集将在生成时自动加载，不过也可以在运行时将特定的程序集加载到当前应用程序域中。有关更多信息，请参见[应用程序域](#)。

没有办法卸载单独的程序集而不卸载包含它的所有应用程序域。即使程序集已在范围之外，实际的程序集文件仍然保持被加载，直至包含它的所有应用程序域都被卸载。

如果想要卸载某些程序集而不卸载其他程序集，可考虑创建新的应用程序域，在该域中执行代码，然后卸载该应用程序域。有关更多信息，请参见[如何：在另一个应用程序域中执行代码](#)。

将程序集加载到应用程序域中

- 使用 `AppDomain` 和 `System.Reflection` 类中包含的几个加载方法之一。有关更多信息，请参见[将程序集加载到应用程序域中](#)。

卸载应用程序域

- 没有办法卸载单独的程序集而不卸载包含它的所有应用程序域。使用 `AppDomain` 中的 `Unload` 方法可卸载应用程序域。有关更多信息，请参见[卸载应用程序域](#)。

请参见

任务

[如何：将程序集加载到应用程序域中](#)

参考

[应用程序域 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[程序集和全局程序集缓存 \(C# 编程指南\)](#)

如何:与其他应用程序共享程序集 (C# 编程指南)

程序集可以是私有的也可以是共享的:默认情况下,大多数简单的 C# 程序都包含一个私有程序集,原因是不打算将该程序集供其他应用程序使用。

为了与其他应用程序共享程序集,必须将该程序集置于[全局程序集缓存 \(GAC\)](#) 中。

共享程序集

1. 创建程序集。有关更多信息,请参见[创建程序集](#)。
2. 为程序集指定一个强名称。有关更多信息,请参见[如何:使用强名称为程序集签名](#)。
3. 为程序集指定版本信息。有关更多信息,请参见[程序集版本控制](#)。
4. 将您的程序集添加到全局程序集缓存中。有关更多信息,请参见[如何:将程序集安装到全局程序集缓存](#)。
5. 从其他应用程序中访问该程序集包含的类型。有关更多信息,请参见[如何:引用具有强名称的程序集](#)。

请参见

概念

[C# 编程指南](#)

[具有强名称的程序集](#)

其他资源

[使用程序集编程](#)

[使用程序集和全局程序集缓存](#)

[创建和使用具有强名称的程序集](#)

属性 (C# 编程指南)

属性提供功能强大的方法以将声明信息与 C# 代码(类型、方法、属性等)相关联。一旦属性与程序实体关联，即可在运行时使用名为[反射](#)的技术对属性进行查询。

属性以两种形式存在：一种是在公共语言运行库的基类库中定义的属性，另一种是可以创建，可以向代码中添加附加信息的自定义属性。此信息可在以后以编程方式检索。

在本例中，属性 [System.Reflection.TypeAttributes.Serializable](#) 用来将特定特性应用于类：

C#

```
[System.Serializable]
public class SampleClass
{
    // Objects of this type can be serialized.
}
```

属性概述

属性具有以下特点：

- 属性可向程序中添加元数据。元数据是嵌入程序中的信息，如编译器指令或数据描述。
- 程序可以使用[反射](#)检查自己的元数据。请参见[使用反射访问属性](#)。
- 通常使用属性与 COM 交互。

相关章节

有关更多信息，请参见：

- [使用属性 \(C# 编程指南\)](#)
- [创建自定义属性 \(C# 编程指南\)](#)
- [消除属性目标的歧义性 \(C# 编程指南\)](#)
- [使用反射访问属性 \(C# 编程指南\)](#)
- [如何：使用属性创建 C/C++ 联合 \(C# 编程指南\)](#)
- [通用属性 \(C# 编程指南\)](#)
- [“属性”示例](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.12 属性
- 17 属性

请参见

概念

[C# 编程指南](#)

[反射 \(C# 编程指南\)](#)

[属性 \(Attribute\) 概述](#)

[特性的常见用途](#)

使用属性 (C# 编程指南)

属性可以放置在几乎所有的声明中(但特定的属性可能限制在其上有效的声明类型)。在语法上, 属性的指定方法为: 将括在方括号中的属性名置于其适用的实体声明之前。例如, 具有 **DllImport** 属性的方法将声明如下:

C#

```
[System.Runtime.InteropServices.DllImport("user32.dll")]
extern static void SampleMethod();
```

有关更多信息, 请参见 [DllImportAttribute 类](#)。

许多属性都有参数, 而这些参数可以是定位(未命名)参数也可以是命名参数。任何定位参数都必须按特定顺序指定并且不能省略, 而命名参数是可选的且可以按任意顺序指定。首先指定定位参数。例如, 这三个属性是等效的:

```
[DllImport("user32.dll")]
[DllImport("user32.dll", SetLastError=false, ExactSpelling=false)]
[DllImport("user32.dll", ExactSpelling=false, SetLastError=false)]
```

第一个参数(DLL 名称)是定位参数并且总是第一个出现, 其他参数为命名参数。在这种情况下, 两个命名参数均默认为 false, 因此可将其省略。有关默认参数值的信息, 请参考各个属性的文档。

在一个声明中可以放置多个属性, 可分开放置, 也可放在同一组括号中:

C#

```
void MethodA([In][Out] ref double x) { }
void MethodB([Out][In] ref double x) { }
void MethodC([In, Out] ref double x) { }
```

某些属性对于给定实体可以指定多次。例如, [Conditional](#) 就是一个可多次使用的属性:

C#

```
[Conditional("DEBUG"), Conditional("TEST1")]
void TraceMethod()
{
    // ...
}
```

注意

根据约定, 所有属性名称都以单词“Attribute”结束, 以便将它们与“.NET Framework”中的其他项区分。但是, 在代码中使用属性时不需要指定属性后缀。例如, `[DllImport]` 虽等效于 `[DllImportAttribute]`, 但 `DllImportAttribute` 才是该属性在 .NET Framework 中的实际名称。

请参见

参考

[消除属性目标的歧义性 \(C# 编程指南\)](#)

[创建自定义属性 \(C# 编程指南\)](#)

[使用反射访问属性 \(C# 编程指南\)](#)

[Attribute](#)

[System.Reflection](#)

[概念](#)

[C# 编程指南](#)

[反射 \(C# 编程指南\)](#)

[属性 \(C# 编程指南\)](#)

创建自定义属性(C# 编程指南)

通过定义一个属性类，可以创建您自己的自定义属性。该属性类直接或间接地从 [Attribute](#) 派生，有助于方便快捷地在元数据中标识属性定义。假设您要用编写类或结构的程序员的名字标记类和结构。可以定义一个自定义 `Author` 属性类：

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
                      System.AttributeTargets.Struct)]
public class Author : System.Attribute
{
    private string name;
    public double version;

    public Author(string name)
    {
        this.name = name;
        version = 1.0;
    }
}
```

类名是属性名 `Author`。它由 **System.Attribute** 派生而来，因此是自定义属性类。构造函数的参数是自定义属性的定位参数（在本例中为 `name`），任何公共读写字段或属性都是命名参数（在本例中，`version` 是唯一的命名参数）。请注意 **AttributeUsage** 属性的用法，它使得 `Author` 属性仅在 **class** 和 **struct** 声明中有效。

可以按如下所示使用此新属性：

C#

```
[Author("H. Ackerman", version = 1.1)]
class SampleClass
{
    // H. Ackerman's code goes here...
}
```

AttributeUsage 有一个命名参数 **AllowMultiple**，使用它可以使自定义属性成为一次性使用或可以使用多次的属性。

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
                      System.AttributeTargets.Struct,
                      AllowMultiple = true) // multiuse attribute]
public class Author : System.Attribute
```

C#

```
[Author("H. Ackerman", version = 1.1)]
[Author("M. Knott", version = 1.2)]
class SampleClass
{
    // H. Ackerman's code goes here...
    // M. Knott's code goes here...
}
```

请参见

参考

[使用属性\(C# 编程指南\)](#)

[消除属性目标的歧义性\(C# 编程指南\)](#)

[使用反射访问属性\(C# 编程指南\)](#)

[System.Reflection](#)

[概念](#)

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

消除属性目标的歧义性(C# 编程指南)

在某些情况下，属性的目标(即属性适用于的实体)显得不明确。例如，在以下方法声明中，`SomeAttr` 属性可以适用于方法或方法的返回值：

C#

```
public class SomeAttr : System.Attribute { }

[SomeAttr]
int Method()
{
    return 0;
}
```

这种情况在封送处理时经常出现。为解析多义性，C# 对于每种声明都有一组默认目标，而通过显式指定属性目标可重写这些目标。

C#

```
// default: applies to method
[SomeAttr]
int Method1() { return 0; }

// applies to method
[method: SomeAttr]
int Method2() { return 0; }

// applies to return value
[return: SomeAttr]
int Method3() { return 0; }
```

请注意，这与 `SomeAttr` 被定义为有效的目标无关；也就是说，即使 `SomeAttr` 被定义为只适用于返回值，也必须指定 `return` 目标。换言之，编译器将不使用 **AttributeUsage** 信息解析不明确的属性目标。有关更多信息，请参见 [AttributeUsage\(C# 编程指南\)](#)。

属性目标的语法如下：

[target : attribute-list]

参数

target

以下之一：程序集、字段、事件、方法、模块、参数、属性、返回值、类型。

attribute-list

适用属性的列表。

下表列出了允许属性的所有声明。对于每个声明，声明中属性的可能目标在第二列中列出。以粗体显示的目标是默认值。

声明	可能的目标
程序集	assembly
模块	module
类	type
结构	type
接口	type

枚举	type
委托	type 、return
方法	method 、return
参数	param
字段	field
属性 — 索引器	property
属性 — get 访问器	method 、return
属性 — set 访问器	method 、param、return
事件 — 字段	event 、field、method
事件 — 属性	event 、property
事件 — 添加	method 、param
事件 — 移除	method 、param

程序集级属性和模块级属性没有默认目标。有关更多信息，请参见[全局属性](#)。

示例

C#

```
using System.Runtime.InteropServices;
```

C#

```
[Guid("12345678-1234-1234-1234-123456789abc"), InterfaceType(ComInterfaceType.InterfaceIsIUnknown)]
interface ISampleInterface
{
    [DispId(17)] // set the DISPID of the method
    [return: MarshalAs(UnmanagedType.Interface)] // set the marshaling on the return type
    object DoWork();
}
```

请参见

参考

[使用属性\(C# 编程指南\)](#)

[创建自定义属性\(C# 编程指南\)](#)

[使用反射访问属性\(C# 编程指南\)](#)

[System.Reflection](#)

[Attribute](#)

概念

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

使用反射访问属性(C# 编程指南)

如果没有检索自定义属性的信息和对其进行操作的方法，则定义自定义属性并将其放置在源代码中就没有意义。C# 具有一个反射系统，可用来检索用自定义属性定义的信息。主要方法是 **GetCustomAttributes**，它返回对象数组，这些对象在运行时等效于源代码属性。此方法具有多个重载版本。有关更多信息，请参见 [Attribute](#)。

属性规范，如：

C#

```
[Author("H. Ackerman", version = 1.1)]
class SampleClass
```

在概念上等效于：

C#

```
Author anonymousAuthorObject = new Author("H. Ackerman");
anonymousAuthorObject.version = 1.1;
```

但是，直到查询 SampleClass 以获取属性时才会执行此代码。对 SampleClass 调用 **GetCustomAttributes** 会导致按上述方式构造并初始化一个 Author 对象。如果类还有其他属性，则其他属性对象的以类似方式构造。然后 **GetCustomAttributes** 返回 Author 对象和数组中的任何其他属性对象。之后就可以对此数组进行迭代，确定根据每个数组元素的类型所应用的属性，并从属性对象中提取信息。

示例

下面是一个完整的示例。定义一个自定义属性，将其应用于若干实体并通过反射进行检索。

C#

```
[System.AttributeUsage(System.AttributeTargets.Class |
    System.AttributeTargets.Struct,
    AllowMultiple = true) // multiuse attribute
]
public class Author : System.Attribute
{
    string name;
    public double version;

    public Author(string name)
    {
        this.name = name;
        version = 1.0; // Default value
    }

    public string GetName()
    {
        return name;
    }
}

[Author("H. Ackerman")]
private class FirstClass
{
    // ...
}

// No Author attribute
private class SecondClass
{
    // ...
}
```

```
[Author("H. Ackerman"), Author("M. Knott", version = 2.0)]
private class ThirdClass
{
    // ...
}

class TestAuthorAttribute
{
    static void Main()
    {
        PrintAuthorInfo(typeof(FirstClass));
        PrintAuthorInfo(typeof(SecondClass));
        PrintAuthorInfo(typeof(ThirdClass));
    }

    private static void PrintAuthorInfo(System.Type t)
    {
        System.Console.WriteLine("Author information for {0}", t);
        System.Attribute[] attrs = System.Attribute.GetCustomAttributes(t); // reflection

        foreach (System.Attribute attr in attrs)
        {
            if (attr is Author)
            {
                Author a = (Author)attr;
                System.Console.WriteLine("    {0}, version {1:f}", a.GetName(), a.version);
            }
        }
    }
}
```

输出

```
Author information for FirstClass
H. Ackerman, version 1.00
Author information for SecondClass
Author information for ThirdClass
H. Ackerman, version 1.00
M. Knott, version 2.00
```

请参见

参考

[使用属性\(C# 编程指南\)](#)

[消除属性目标的歧义性\(C# 编程指南\)](#)

[创建自定义属性\(C# 编程指南\)](#)

[System.Reflection](#)

[Attribute](#)

概念

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

如何：使用属性创建 C/C++ 联合 (C# 编程指南)

通过使用属性可以自定义结构在内存中的布局方式。例如，可以使用 `StructLayout(LayoutKind.Explicit)` 和 `FieldOffset` 属性创建在 C/C++ 中称为联合的布局。

示例

在上一个代码段中，`TestUnion` 的所有字段都从内存中的同一位置开始。

C#

```
[System.Runtime.InteropServices.StructLayout(LayoutKind.Explicit)]
struct TestUnion
{
    [System.Runtime.InteropServices.FieldOffset(0)]
    public int i;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public double d;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public char c;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public byte b;
}
```

以下是字段从其他显式设置的位置开始的另一个示例。

C#

```
[System.Runtime.InteropServices.StructLayout(LayoutKind.Explicit)]
struct TestExplicit
{
    [System.Runtime.InteropServices.FieldOffset(0)]
    public long lg;

    [System.Runtime.InteropServices.FieldOffset(0)]
    public int i1;

    [System.Runtime.InteropServices.FieldOffset(4)]
    public int i2;

    [System.Runtime.InteropServices.FieldOffset(8)]
    public double d;

    [System.Runtime.InteropServices.FieldOffset(12)]
    public char c;

    [System.Runtime.InteropServices.FieldOffset(14)]
    public byte b;
}
```

`i1` 和 `i2` 这两个 `int` 字段共享与 `lg` 相同的内存位置。使用平台调用时，这种结构布局控制很有用。

请参见

参考

[使用属性 \(C# 编程指南\)](#)

[消除属性目标的歧义性 \(C# 编程指南\)](#)

[创建自定义属性 \(C# 编程指南\)](#)

[使用反射访问属性 \(C# 编程指南\)](#)

[System.Reflection](#)

[Attribute](#)

[概念](#)

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

通用属性 (C# 编程指南)

本节描述 C# 程序中最常用的属性。

本节内容

- [Conditional \(C# 编程指南\)](#)
- [Obsolete \(C# 编程指南\)](#)
- [全局属性 \(C# 编程指南\)](#)
- [AttributeUsage \(C# 编程指南\)](#)

请参见

参考

[使用属性 \(C# 编程指南\)](#)

[使用反射访问属性 \(C# 编程指南\)](#)

[System.Reflection](#)

[Attribute](#)

概念

[C# 编程指南](#)

[反射 \(C# 编程指南\)](#)

Conditional(C# 编程指南)

根据预处理标识符执行方法。Conditional 属性是 [ConditionalAttribute](#) 的别名，可应用于方法或属性类。

在本示例中，Conditional 应用于方法以启用或禁用程序特定的诊断信息的显示：

```
#define TRACE_ON
using System;
using System.Diagnostics;

public class Trace
{
    [Conditional("TRACE_ON")]
    public static void Msg(string msg)
    {
        Console.WriteLine(msg);
    }
}

public class ProgramClass
{
    static void Main()
    {
        Trace.Msg("Now in Main...");
        Console.WriteLine("Done.");
    }
}
```

如果未定义 TRACE_ON 标识符，则将不会显示跟踪输出。

Conditional 属性经常与 DEBUG 标识符一起使用以启用调试版本的跟踪和日志记录功能（在发布版本中没有这两种功能），如下例所示：

```
[Conditional("DEBUG")]
static void DebugMethod()
{
}
```

备注

当调用标记为条件的方法时，指定的预处理符号的存在或不存在决定是否包含或省略此调用。如果定义了该符号，则包含调用；否则省略调用。使用 Conditional 是封闭 #if 和 #endif 内部方法的替代方法，它更整洁、更别致、减少了出错的机会，如下例所示：

```
#if DEBUG
void ConditionalMethod()
{
}
#endif
```

条件方法必须是类或结构声明中的方法，而且必须具有 `void` 返回类型。

使用多个标识符

如果某个方法具有多个 **Conditional** 属性，且至少定义了多个条件符号（换言之，这些符号彼此之间存在逻辑“或”关系）中的一个，则将包含对该方法的调用。在本例中，`A` 或 `B` 的存在将导致方法调用：

C#

```
[Conditional("A"), Conditional("B")]
static void DoIfAorB()
{}
```

```
// ...  
}
```

若要获得对符号进行逻辑“与”运算的效果，可以定义序列条件方法。例如，仅当 A 和 B 均已定义时，才能执行下面的第二种方法：

C#

```
[Conditional("A")]
static void DoIfA()
{
    DoIfAandB();
}

[Conditional("B")]
static void DoIfAandB()
{
    // Code to execute when both A and B are defined...
}
```

使用具有属性类的条件

Conditional 属性还可被应用于属性类定义。在本例中，仅当定义了 DEBUG 时，自定义属性 Documentation 才向元数据添加信息。

C#

```
[Conditional("DEBUG")]
public class Documentation : System.Attribute
{
    string text;

    public Documentation(string text)
    {
        this.text = text;
    }
}

class SampleClass
{
    // This attribute will only be included if DEBUG is defined.
    [Documentation("This method displays an integer.")]
    static void DoWork(int i)
    {
        System.Console.WriteLine(i.ToString());
    }
}
```

请参见

参考

[消除属性目标的歧义性\(C# 编程指南\)](#)

[创建自定义属性\(C# 编程指南\)](#)

[使用反射访问属性\(C# 编程指南\)](#)

[System.Reflection](#)

[Attribute](#)

概念

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

Obsolete(C# 编程指南)

`Obsolete` 属性将某个程序实体标记为一个建议不再使用的实体。每次使用被标记为已过时的实体时，随后将生成警告或错误，这取决于属性是如何配置的。例如：

```
[System.Obsolete("use class B")]
class A
{
    public void Method() { }
}
class B
{
    [System.Obsolete("use NewMethod", true)]
    public void OldMethod() { }
    public void NewMethod() { }
}
```

在此例中，`Obsolete` 属性应用于类 `A` 和方法 `B.OldMethod`。由于应用于 `B.OldMethod` 的属性构造函数的第二个参数设置为 `true`，因此使用此方法将导致编译器错误，而使用类 `A` 只会产生警告。但是，调用 `B.NewMethod` 既不产生警告也不产生错误。

向属性构造函数提供的作为第一个参数的字符串将显示为警告或错误的一部分。例如，当与前面的定义一起使用时，下面的代码将生成两个警告和一个错误：

```
// Generates 2 warnings:
A a = new A();
// Generate no errors or warnings:
B b = new B();
b.NewMethod();
// Generates an error, terminating compilation:
b.OldMethod();
```

为类 `A` 产生两个警告：一个用于声明类引用，一个用于类构造函数。

可在不使用参数的情况下使用 `Obsolete` 属性，但要包括此项已过时的原因及改用什么项的建议。

`Obsolete` 属性是一个单用途属性，并且可应用于允许属性的任何实体。`Obsolete` 是 [ObsoleteAttribute](#) 的别名。

请参见

参考

[消除属性目标的歧义性\(C# 编程指南\)](#)

[创建自定义属性\(C# 编程指南\)](#)

[使用反射访问属性\(C# 编程指南\)](#)

Attribute

[System.Reflection](#)

概念

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

全局属性 (C# 编程指南)

大多数属性适用于特定的语言元素，如类或方法；但是有些属性是全局的，它们适用于整个程序集或模块。例如，[AssemblyVersionAttribute](#) 属性可用于向程序集中嵌入版本信息，如下例所示：

```
[assembly: AssemblyVersion("1.0.0.0")]
```

全局属性在源代码中出现在任何顶级 `using` 指令之后以及任何类型或命名空间声明之前。全局属性可显示在多个源文件中，但这些文件必须在单一编译传递中编译。

以下是一些常用的 .NET Framework 程序集级的属性：

[AssemblyCompanyAttribute](#)

[AssemblyConfigurationAttribute](#)

[AssemblyCopyrightAttribute](#)

[AssemblyCultureAttribute](#)

[AssemblyDescriptionAttribute](#)

[AssemblyProductAttribute](#)

[AssemblyTitleAttribute](#)

[AssemblyTrademarkAttribute](#)

这些属性用于基于 Visual Studio Windows 窗体应用程序模板的项目中。此模板包含一个名为 `AssemblyInfo.cs` 的文件，该文件包括这些属性实例化：

```
[assembly: AssemblyTitle("WindowsApplication1")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("Microsoft")]
[assembly: AssemblyProduct("WindowsApplication1")]
[assembly: AssemblyCopyright("Copyright © Microsoft 2005")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
```

注意

如果不是正在创建程序集，则将忽略程序集级属性。

程序集签名属性

在 Visual Studio 的早期版本中，使用这些程序集级属性执行具有强名称的签名程序集

- [AssemblyKeyFileAttribute](#)
- [AssemblyKeyNameAttribute](#)
- [AssemblyDelaySignAttribute](#)

现在仍可支持这样做，但是签名程序集的首选方法是使用项目设计器中的签名页。有关更多信息，请参见“[项目设计器”->“签名”页和如何：对程序集进行签名 \(Visual Studio\)](#)”。

请参见

参考

[创建自定义属性 \(C# 编程指南\)](#)

[使用反射访问属性 \(C# 编程指南\)](#)

[System.Reflection](#)

概念

[C# 编程指南](#)

[属性 \(C# 编程指南\)](#)

[其他资源](#)

AttributeUsage(C# 编程指南)

确定可以如何使用自定义属性类。AttributeUsage 是一个可应用于自定义属性定义的属性，自定义属性定义来控制如何应用新属性。显式应用的默认设置与此类似：

```
[System.AttributeUsage(System.AttributeTargets.All,
    AllowMultiple=false,
    Inherited=true)]
class NewAttribute : System.Attribute { }
```

在本例中，NewAttribute 类可应用于任何支持属性的代码实体，但是对每个实体只可应用一次。当应用于基类时，它可由派生类继承。

AllowMultiple 和 Inherited 参数是可选的，所以此代码具有相同的效果：

```
[System.AttributeUsage(System.AttributeTargets.All)]
class NewAttribute : System.Attribute { }
```

第一个 AttributeUsage 参数必须是 [AttributeTargets 枚举的一个或多个元素](#)。多个目标类型可同时进行“或”运算，如下例所示：

```
using System;
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Field)]
class NewPropertyOrFieldAttribute : Attribute { }
```

如果 AllowMultiple 参数设置为 **true**，则返回属性可对单个实体应用多次，如下例所示：

```
using System;
[AttributeUsage(AttributeTargets.Class, AllowMultiple=true)]
class MultiUseAttr : Attribute { }

[MultiUseAttr][MultiUseAttr]
class Class1 { }

[MultiUseAttr, MultiUseAttr]
class Class2 { }
```

在这种情况下，由于 AllowMultiple 设置为 **true**，MultiUseAttr 可反复应用。所示的应用多个属性的这两种格式均有效。

如果 Inherited 设置为 **false**，则该属性不由从属性化的类派生的类继承。例如：

```
using System;
[AttributeUsage(AttributeTargets.Class, Inherited=false)]
class Attr1 : Attribute { }

[Attr1]
class BClass { }

class DClass : BClass { }
```

在这种情况下，Attr1 不通过继承应用于 DClass。

备注

AttributeUsage 属性是一个单用途属性 — 它无法对相同的类应用多次。AttributeUsage 是 [AttributeUsageAttribute](#) 的别名。

有关更多信息，请参见[使用反射访问属性\(C# 编程指南\)](#)。

示例

下面的示例将阐释 Inherited 参数和 AllowMultiple 参数对 AttributeUsage 属性的效果，以及如何才能枚举应用于类的自定义属性。

```

using System;

// Create some custom attributes:
[AttributeUsage(System.AttributeTargets.Class, Inherited=false)]
class A1 : System.Attribute { }

[AttributeUsage(System.AttributeTargets.Class)]
class A2 : System.Attribute { }

[AttributeUsage(System.AttributeTargets.Class, AllowMultiple=true)]
class A3 : System.Attribute { }

// Apply custom attributes to classes:
[A1,A2]
class BaseClass { }

[A3,A3]
class DerivedClass : BaseClass { }

public class TestAttributeUsage
{
    static void Main()
    {
        BaseClass b = new BaseClass();
        DerivedClass d = new DerivedClass();

        // Display custom attributes for each class.
        Console.WriteLine("Attributes on Base Class:");
        object[] attrs = b.GetType().GetCustomAttributes(true);
        foreach (Attribute attr in attrs)
        {
            Console.WriteLine(attr);
        }

        Console.WriteLine("Attributes on Derived Class:");
        attrs = d.GetType().GetCustomAttributes(true);
        foreach (Attribute attr in attrs)
        {
            Console.WriteLine(attr);
        }
    }
}

```

示例输出

```

Attributes on Base Class:
A1
A2
Attributes on Derived Class:
A3
A3
A2

```

请参见

参考

[使用属性\(C# 编程指南\)](#)

[消除属性目标的歧义性\(C# 编程指南\)](#)

[创建自定义属性\(C# 编程指南\)](#)

[使用反射访问属性\(C# 编程指南\)](#)

[Attribute](#)

[System.Reflection](#)

概念

[C# 编程指南](#)

[反射\(C# 编程指南\)](#)

[属性\(C# 编程指南\)](#)

集合类(C# 编程指南)

.NET Framework 提供了用于数据存储和检索的专用类。这些类提供对堆栈、队列、列表和哈希表的支持。大多数集合类实现相同的接口，可继承这些接口来创建适应更为专业的数据存储需要的新集合类。

注意

针对 .NET Framework 的 2.0 版和更高版本的应用程序应当使用 [System.Collections.Generic](#) 命名空间中的泛型集合类，与对应的非泛型类相比，这些类提供了更高的类型安全性和效率。

C#

```
ArrayList list = new ArrayList();
list.Add(10);
list.Add(20);
```

集合类概述

集合类具有以下特点：

- 集合类定义为 [System.Collections](#) 或 [System.Collections.Generic](#) 命名空间的一部分。
- 大多数集合类都派生自 **ICollection**、**IComparer**、**IEnumerable**、**IList**、**IDictionary** 和 **IDictionaryEnumerator** 接口以及它们的等效泛型接口。
- 使用泛型集合类可以提供更高的类型安全性，在某些情况下还可以提供更好的性能，尤其是在存储值类型时，这些优势会体现得更明显。有关更多信息，请参见[泛型的优点](#)。

相关章节

- [何时使用泛型集合](#)
- [集合和数据结构](#)
- [选择集合类](#)
- [集合内的比较和排序](#)
- [创建和操作集合](#)
- [如何：使用 foreach 访问集合类\(C# 编程指南\)](#)
- [“集合类”示例](#)

请参见

概念

[C# 编程指南](#)

[数组\(C# 编程指南\)](#)

如何：使用 **foreach** 访问集合类(C# 编程指南)

下面的代码示例阐释如何编写可与 **foreach** 一起使用的非泛型集合类。该类是字符串标记化拆分器，类似于 C 运行时库函数 **strtok**。

注意

此示例描述的是只有在您无法使用泛型集合类时才采用的推荐做法。C# 语言和 .NET Framework 的 2.0 版和更高版本支持泛型。要通过示例来了解如何实现支持 **IEnumerable<T>** (因此避免了下面讨论的问题) 的类型安全泛型集合类，请参见[如何：为泛型列表创建迭代器块\(C# 编程指南\)](#)。

在下面的示例中，`Tokens` 使用“ ”和“-”作为分隔符将句子“This is a sample sentence.”拆分为标记，并使用 **foreach** 语句枚举这些标记：

C#

```
Tokens f = new Tokens("This is a sample sentence.", new char[] {' ', '-'});

foreach (string item in f)
{
    System.Console.WriteLine(item);
}
```

`Tokens` 在内部使用一个数组，该数组自行实现 **IEnumerator** 和 **IEnumerable**。该代码示例本可以利用数组本身的枚举方法，但那样会使本示例的用意无法体现出来。

在 C# 中，集合类并非必须严格从 **IEnumerable** 和 **IEnumerator** 继承才能与 **foreach** 兼容；只要类有所需的 **GetEnumerator**、**MoveNext**、**Reset** 和 **Current** 成员，便可以与 **foreach** 一起使用。省略接口的好处为，使您可以将 **Current** 的返回类型定义得比 **object** 更明确，从而提供了类型安全。

例如，从上面的示例代码开始，更改以下几行：

```
// No longer inherits from IEnumerable:
public class Tokens
// Doesn't return an IEnumerator:
public TokenEnumerator GetEnumerator()
// No longer inherits from IEnumerator:
public class TokenEnumerator
// Type-safe: returns string, not object:
public string Current
```

现在，由于 `Current` 返回字符串，因此当 **foreach** 语句中使用了不兼容的类型时，编译器便能够检测到：

```
// Error: cannot convert string to int:
foreach (int item in f)
```

省略 **IEnumerable** 和 **IEnumerator** 的缺点是，集合类不再能够与其他公共语言运行库兼容语言的 **foreach** 语句或等效项交互操作。

您可以同时拥有两者的优点，即 C# 中的类型安全以及与其他公共语言运行库兼容语言的互操作性，方法是从 **IEnumerable** 和 **IEnumerator** 继承并使用显式接口实现，如下面的示例所示。

示例

C#

```
using System.Collections;

// Declare the Tokens class:
public class Tokens : IEnumerable
{
    private string[] elements;
```

```

Tokens(string source, char[] delimiters)
{
    // Parse the string into tokens:
    elements = source.Split(delimiters);
}

// IEnumerable Interface Implementation:
// Declaration of the GetEnumerator() method
// required by IEnumerable
public IEnumerator GetEnumerator()
{
    return new TokenEnumerator(this);
}

// Inner class implements IEnumerator interface:
private class TokenEnumerator : IEnumerator
{
    private int position = -1;
    private Tokens t;

    public TokenEnumerator(Tokens t)
    {
        this.t = t;
    }

    // Declare the MoveNext method required by IEnumerator:
    public bool MoveNext()
    {
        if (position < t.elements.Length - 1)
        {
            position++;
            return true;
        }
        else
        {
            return false;
        }
    }

    // Declare the Reset method required by IEnumerator:
    public void Reset()
    {
        position = -1;
    }

    // Declare the Current property required by IEnumerator:
    public object Current
    {
        get
        {
            return t.elements[position];
        }
    }
}

// Test Tokens, TokenEnumerator
static void Main()
{
    // Testing Tokens by breaking the string into tokens:
    Tokens f = new Tokens("This is a sample sentence.", new char[] { ' ', '-' });

    foreach (string item in f)
    {
        System.Console.WriteLine(item);
    }
}

```

```
    }  
}
```

输出

```
This  
is  
a  
sample  
sentence.
```

请参见

参考

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

[集合类 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

异常和异常处理(C# 编程指南)

C# 语言的异常处理功能提供了处理程序运行时出现的任何意外或异常情况的方法。异常处理使用 **try**、**catch** 和 **finally** 关键字来尝试可能未成功的操作，处理失败，以及在事后清理资源。异常可以由公共语言运行库 (CLR)、第三方库或使用 **throw** 关键字的应用程序代码生成。

此示例中使用一个方法检测是否有被零除的情况；如果有，则捕获该错误。如果没有异常处理，此程序将终止并产生“DivideByZeroException 未处理”错误。

```
int SafeDivision(int x, int y)
{
    try
    {
        return (x / y);
    }
    catch (System.DivideByZeroException dbz)
    {
        System.Console.WriteLine("Division by zero attempted!");
        return 0;
    }
}
```

异常概述

异常具有以下特点：

- 在应用程序遇到异常情况(如被零除情况或内存不足警告)时，就会产生异常。
- 在可能引发异常的语句周围使用 **try** 块。
- **try** 块发生异常后，控制流会立即跳转到关联的异常处理程序(如果存在)。
- 如果给定异常没有异常处理程序，则程序将停止执行，并显示一条错误消息。
- 如果 **catch** 块定义了一个异常变量，则可以使用它来获取有关所发生异常的类型的更多信息。
- 可能导致异常的操作通过 **try** 关键字来执行。
- 异常处理程序是在异常发生时执行的代码块。在 C# 中，**catch** 关键字用于定义异常处理程序。
- 程序可以使用 **throw** 关键字显式地引发异常。
- 异常对象包含有关错误的详细信息，其中包括调用堆栈的状态以及有关错误的文本说明。
- 即使引发了异常，**finally** 块中的代码也会执行，从而使程序可以释放资源。

相关章节

有关异常和异常处理的更多信息，请参见以下主题：

- [使用异常](#)
- [处理异常](#)
- [创建和引发异常](#)
- [编译器生成的异常](#)
- [如何：使用 Try/Catch 处理异常](#)
- [如何：使用 Try/Finally 执行清理代码](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [8.9.5 throw 语句](#)
- [8.10 try 语句](#)

- 16 异常

请参见

参考

[C# 关键字](#)

[throw\(C# 参考\)](#)

[try-catch\(C# 参考\)](#)

[try-finally\(C# 参考\)](#)

[try-catch-finally\(C# 参考\)](#)

概念

[C# 编程指南](#)

[异常概述](#)

[其他资源](#)

[异常设计准则](#)

[处理和引发异常](#)

使用异常 (C# 编程指南)

在 C# 中，程序中的运行时错误使用一种称为“异常”的机制在程序中传播。异常由遇到错误的代码引发，由能够更正错误的代码捕捉。异常可由 .NET Framework 公共语言运行库 (CLR) 或由程序中的代码引发。一旦引发了一个异常，这个异常就会在调用堆栈中往上传播，直到找到针对它的 **catch** 语句。未捕获的异常由系统提供的通用异常处理程序处理，该处理程序会显示一个对话框。

异常由从 [Exception](#) 派生的类表示。此类标识异常的类型，包含详细描述异常的属性。引发异常涉及到创建一个异常派生类的实例，配置异常的属性（可选），然后使用 **throw** 关键字引发该对象。例如：

```
C#
private static void TestThrow()
{
    System.ApplicationException ex =
        new System.ApplicationException("Demonstration exception in TestThrow()");
    throw ex;
}
```

在引发异常之后，运行库检查当前语句以确定它是否在 **try** 块中。如果是，则检查与该 **try** 块关联的任何 **catch** 块，以确定它们是否能够捕获该异常。**Catch** 块通常会指定异常类型；如果该 **catch** 块的类型与异常或其基类的类型相同，则该 **catch** 块就能够处理该方法。例如：

```
C#
static void TestCatch()
{
    try
    {
        TestThrow();
    }
    catch (System.ApplicationException ex)
    {
        System.Console.WriteLine(ex.ToString());
    }
}
```

如果引发异常的语句不在 **try** 块中，或者包含该语句的 **try** 块没有匹配的 **catch** 块，运行库将检查调用方法中是否有 **try** 语句和 **catch** 块。运行库将在调用堆栈中向上继续搜索兼容的 **catch** 块。在找到并执行 **catch** 块之后，控制权将传递给 **catch** 块之后的第一个语句。

一个 **try** 语句可能包含多个 **catch** 块。将执行第一个能够处理该异常的 **catch** 语句；任何后续的 **catch** 语句都将被忽略，即使它们是兼容的也如此。例如：

```
C#
static void TestCatch2()
{
    try
    {
        TestThrow();
    }
    catch (System.ApplicationException ex)
    {
        System.Console.WriteLine(ex.ToString()); // this block will be executed
    }
    catch (System.Exception ex)
    {
        System.Console.WriteLine(ex.ToString()); // this block will NOT be executed
    }

    System.Console.WriteLine("Done"); // this statement is executed after the catch block
}
```

```
}
```

在执行 **catch** 块之前，将检查由运行库评估过的 **try** 块（包括包含兼容的 **catch** 块的 **try** 块）中是否有 **finally** 块。**Finally** 块允许程序员清理中止的 **try** 块可能留下的任何不明确状态，或释放任何外部资源（如图形句柄、数据库连接或文件流），而不用等待运行库中的垃圾回收器来完成这些对象。例如：

C#

```
static void TestFinally()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = new System.IO.FileInfo("C:\\file.txt");

    try
    {
        file = fileInfo.OpenWrite();
        file.WriteByte(0xF);
    }
    finally
    {
        // Closing the file allows you to reopen it immediately - otherwise IOException is
        // thrown.
        if (file != null)
        {
            file.Close();
        }
    }

    try
    {
        file = fileInfo.OpenWrite();
        System.Console.WriteLine("OpenWrite() succeeded");
    }
    catch (System.IO.IOException)
    {
        System.Console.WriteLine("OpenWrite() failed");
    }
}
```

如果 `WriteByte()` 引发了异常，那么在没有调用 `file.Close()` 的情况下，第二个 **try** 块中尝试重新打开文件的代码就会失败，并且文件将保持锁定状态。由于无论是否引发异常都要执行 **finally** 块，前一示例中的 **finally** 块允许正确地关闭文件并可帮助避免错误。

如果在引发异常之后没有在调用堆栈上找到兼容的 **catch** 块，则会出现三种情况中的一种：

- 如果异常出现在析构函数中，则中止该析构函数并调用基析构函数（如果有）。
- 如果调用堆栈包含静态构造函数或静态字段初始值设定项，则引发一个 `TypeInitializationException`，并将原始异常分配给新异常的 `InnerException` 属性。
- 如果到达线程的开头，则终止线程。

请参见

参考

[异常和异常处理\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

异常处理(C# 编程指南)

C# 程序员使用 `try` 块来对可能受异常影响的代码进行分区，并使用 `catch` 块来处理所产生的任何异常。不管是否引发异常，都可以使用 `finally` 块来执行代码；有时必须这样做，因为如果引发了异常，将不执行 `try/catch` 构造后面的代码。`try` 块必须与 `catch` 或 `finally` 块一起使用，并且可以包括多个 `catch` 块。例如：

C#

```
try
{
    // Code to try here.
}
catch (System.Exception ex)
{
    // Code to handle exception here.
}
```

C#

```
try
{
    // Code to try here.
}
finally
{
    // Code to execute after try here.
}
```

C#

```
try
{
    // Code to try here.
}
catch (System.Exception ex)
{
    // Code to handle exception here.
}
finally
{
    // Code to execute after try (and possibly catch) here.
}
```

没有 `catch` 或 `finally` 块的 `try` 语句将产生编译器错误。

Catch 块

`catch` 块可以指定要捕捉的异常类型。这个类型称为“异常筛选器”，它必须是 `Exception` 类型，或者必须从此类型派生。应用程序定义的异常应当从 `ApplicationException` 派生。

具有不同异常筛选器的多个 `catch` 块可以串联在一起。多个 `catch` 块的计算顺序是从顶部到底部，但是，对于所引发的每个异常，都只执行一个 `catch` 块。与所引发异常的准确类型或其基类最为匹配的第一个 `catch` 块将被执行。如果没有任何 `catch` 块指定匹配的异常筛选器，那么将执行没有筛选器的 `catch` 块（如果有的话）。重要的是将具有最具体的（派生程度最高的）异常类的 `catch` 块放在最前面。

应捕捉异常的情况：

- 对引发异常的原因有具体的了解，并可实现特定的恢复，例如，捕捉 `FileNotFoundException` 对象并提示用户输入新的文件名。
- 可以新建一个更具体的异常并引发该异常。例如：

C#

```
int GetInt(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch(System.IndexOutOfRangeException e)
    {
        throw new System.ArgumentOutOfRangeException(
            "Parameter index is out of range.");
    }
}
```

- 部分处理异常。例如，**catch** 块可以用来向错误日志中添加项，但在随后重新引发异常，以便允许对异常进行后续处理。例如：

C#

```
try
{
    // try to access a resource
}
catch (System.UnauthorizedAccessException e)
{
    LogError(e); // call a custom error logging procedure
    throw e;      // re-throw the error
}
```

Finally 块

finally 块允许清理在 **try** 块中执行的操作。如果存在 **finally** 块，它将在执行完 **try** 和 **catch** 块之后执行。**finally** 块始终会执行，而与是否引发异常或者是否找到与异常类型匹配的 **catch** 块无关。

可以使用 **finally** 块释放资源(如文件流、数据库连接和图形句柄)，而不用等待由运行库中的垃圾回收器来完成对象。有关更多信息，请参见 [using 语句\(C# 参考\)](#)。

在此示例中，**finally** 块用来关闭在 **try** 块中打开的文件。注意，在关闭文件句柄之前要检查它的状态。如果 **try** 块未能打开文件，则文件句柄仍将设置为 **null**。或者，如果成功打开文件并且没有引发异常，则 **finally** 块仍将执行，并且将关闭打开的文件。

C#

```
System.IO.FileStream file = null;
System.IO.FileInfo fileInfo = new System.IO.FileInfo("C:\\file.txt");
try
{
    file = fileInfo.OpenWrite();
    file.WriteByte(0xF);
}
finally
{
    // check for null because OpenWrite
    // might have failed
    if (file != null)
    {
        file.Close();
    }
}
```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 16 异常
- 8.9.5 throw 语句
- 8.10 try 语句

请参见

参考

[异常和异常处理\(C# 编程指南\)](#)

[try-catch\(C# 参考\)](#)

[try-finally\(C# 参考\)](#)

[try-catch-finally\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

创建和引发异常(C# 编程指南)

异常用于指示在运行程序时发生了错误。此时将创建一个描述错误的异常对象，然后使用 `throw` 关键字“引发”该对象。然后运行库搜索最兼容的异常处理程序。

程序员应在下列情况下引发异常：

- 方法无法完成其中定义的功能。例如，如果方法的参数具有无效值：

C#

```
static void CopyObject(SampleClass original)
{
    if (original == null)
    {
        throw new System.ArgumentException("Parameter cannot be null", "original");
    }
}
```

- 根据对象的状态，对某个对象进行不适当的调用。例如，尝试对只读文件执行写操作。在对象状态不允许某项操作的情况下，引发 `InvalidOperationException` 的一个实例或基于此类的派生类的对象。以下为引发 `InvalidOperationException` 对象的方法的示例：

C#

```
class ProgramLog
{
    System.IO.FileStream logFile = null;
    void OpenLog(System.IO.FileInfo fileName, System.IO.FileMode mode) {}

    void WriteLog()
    {
        if (!this.logFile.CanWrite)
        {
            throw new System.InvalidOperationException("Logfile cannot be read-only");
        }
        // Else write data to the log and return.
    }
}
```

- 方法的参数导致了异常。在此情况下，应捕获原始异常并创建一个 `ArgumentException` 实例。原始异常应作为 `InnerException` 参数传递给 `ArgumentException` 的构造函数：

C#

```
static int GetValueFromArray(int[] array, int index)
{
    try
    {
        return array[index];
    }
    catch (System.IndexOutOfRangeException ex)
    {
        System.ArgumentException argEx = new System.ArgumentException("Index is out of
range", "index", ex);
        throw argEx;
    }
}
```

}

异常包含一个名为 **StackTrace** 的属性，此字符串包含当前调用堆栈上的方法的名称，以及为每个方法引发异常的位置(文件名和行号)。**StackTrace** 对象由 CLR 从 **throw** 语句点开始自动创建，因此必须从堆栈跟踪的开始点引发异常。

所有异常都包含一个名为 **Message** 的属性，应该设置此字符串来解释发生异常的原因。注意，不应将安全敏感信息放在消息文本中。除 **Message** 之外，**ArgumentException** 还包含一个名为 **ParamName** 的属性，应将该属性设置为导致引发异常的参数的名称。对于属性设置器，**ParamName** 应设置为 **value**。

公共的受保护方法应在其无法完成预期功能时引发异常。引发的异常类应该是符合错误条件的最确切的可用异常。这些异常应编写为类功能的一部分，派生类或对原始类的更新应保留相同的行为，以实现向后兼容性。

不应作为正常执行的一部分来使用异常改变程序的流程，异常只能用于报告和处理错误情况。只能引发异常，而不能作为返回值或参数返回异常。程序员不应有意引发 **System.Exception**、**System.SystemException**、**NullReferenceException** 或 **IndexOutOfRangeException**。

定义异常类

程序可以引发 **System** 命名空间中的任何预定义异常类(前面注明的情况除外)，或通过从 **ApplicationException** 派生来创建它们自己的异常类。派生类至少应定义四个构造函数：一个是默认构造函数，一个用来设置消息属性，一个用来设置 **Message** 属性和 **InnerException** 属性。第四个构造函数则用于序列化异常，新的异常类应是可序列化的。例如：

C#

```
public class InvalidDepartmentException : System.ApplicationException
{
    public InvalidDepartmentException() {}
    public InvalidDepartmentException(string message) {}
    public InvalidDepartmentException(string message, System.Exception inner) {}

    // Constructor needed for serialization
    // when exception propagates from a remoting server to the client.
    protected InvalidDepartmentException(System.Runtime.Serialization.SerializationInfo info,
                                         System.Runtime.Serialization.StreamingContext context) {}
}
```

仅当新属性提供的数据有助于解决异常时，才应将其添加到异常类。如果向派生的异常类添加了新属性，则应重写 **ToString()** 以返回添加的信息。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 8.9.5 **throw** 语句
- 8.10 **try** 语句
- 16 异常

请参见

参考

[异常和异常处理\(C# 编程指南\)](#)

[异常处理\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

编译器生成的异常(C# 编程指南)

有些异常作为失败的基本操作的结果由 .NET Framework 的公共语言运行库 (CLR) 自动引发。这些异常及其错误条件列举如下。

异常	说明
ArithmeticException	在算术运算期间发生的异常(如 DivideByZeroException 和 OverflowException)的基类。
ArrayTypeMismatchException	当数组存储给定的元素时, 如果由于该元素的实际类型与数组的实际类型不兼容而导致存储失败, 就会引发此异常。
DivideByZeroException	在试图用零除整数值时引发。
IndexOutOfRangeException	在试图为数组设置小于零或超出数组界限的索引时引发。
InvalidCastException	当从基类型到接口或派生类型的显式转换在运行时失败时, 就会引发此异常。
NullReferenceException	在尝试引用值为 <code>null</code> 的对象时引发。
OutOfMemoryException	在使用 <code>new</code> 运算符分配内存的尝试失败时引发。这表明可用于公共语言运行库的内存已耗尽。
OverflowException	在 <code>checked</code> 上下文中的算术运算溢出时引发。
StackOverflowException	当执行堆栈由于具有太多的挂起方法调用而耗尽时, 就会引发此异常;这通常表明存在非常深的递归或无限递归。
TypeInitializationException	在静态构造函数引发异常并且不存在可以捕捉到它的兼容 <code>catch</code> 子句时引发。

请参见

参考

[异常和异常处理\(C# 编程指南\)](#)

[异常处理\(C# 编程指南\)](#)

[try-catch\(C# 参考\)](#)

[try-finally\(C# 参考\)](#)

[try-catch-finally\(C# 参考\)](#)

概念

[C# 编程指南](#)

如何：使用 try/catch 处理异常 (C# 编程指南)

[try-catch](#) 块的用途是捕捉和处理工作代码所生成的异常。有些异常可以在 **catch** 块中处理，问题解决后异常不会再次引发；但更多情况下，唯一能做的是确保引发适当的异常。

示例

在此示例中，[IndexOutOfRangeException](#) 不是最适当的异常：对本方法而言 [ArgumentOutOfRangeException](#) 更恰当些，因为错误是由调用方传入的 `index` 参数导致的。

C#

```
class TestTryCatch
{
    static int GetInt(int[] array, int index)
    {
        try
        {
            return array[index];
        }
        catch (System.IndexOutOfRangeException e) // CS0168
        {
            System.Console.WriteLine(e.Message);
            //set IndexOutOfRangeException to the new exception's InnerException
            throw new System.ArgumentOutOfRangeException("index parameter is out of range."
, e);
        }
    }
}
```

注释

产生异常的代码包括在 **try** 块中。在其后面紧接着添加一个 **catch** 语句，以便在 [IndexOutOfRangeException](#) 发生时对其进行处理。**catch** 块处理 [IndexOutOfRangeException](#)，并引发更适当的 [ArgumentOutOfRangeException](#) 异常。为给调用方提供尽可能多的信息，应考虑将原始异常指定为新异常的 [InnerException](#)。因为 **InnerException** 属性是只读，所以必须在新异常的构造函数中为其赋值。

[请参见](#)

[参考](#)

[异常和异常处理 \(C# 编程指南\)](#)

[异常处理 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

如何：使用 finally 执行清理代码 (C# 编程指南)

finally 语句的目的是确保即使在引发异常的情况下也能立即进行必要的对象(通常是正在占用外部资源的对象)清理。此类清理功能的一个示例是在使用后立即对 [FileStream](#) 调用 [Close](#), 而不是等待公共语言运行库对该对象进行垃圾回收, 如下所示:

C#

```
static void CodeWithoutCleanup()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = new System.IO.FileInfo("C:\\file.txt");

    file = fileInfo.OpenWrite();
    file.WriteByte(0xF);

    file.Close();
}
```

示例

为了将上面的代码转换为 **try-catch-finally** 语句, 需要将清理代码与工作代码分开, 如下所示。

C#

```
static void CodeWithCleanup()
{
    System.IO.FileStream file = null;
    System.IO.FileInfo fileInfo = null;

    try
    {
        fileInfo = new System.IO.FileInfo("C:\\file.txt");

        file = fileInfo.OpenWrite();
        file.WriteByte(0xF);
    }
    catch(System.Exception e)
    {
        System.Console.WriteLine(e.Message);
    }
    finally
    {
        if (file != null)
        {
            file.Close();
        }
    }
}
```

因为在 [OpenWrite\(\)](#) 调用前, **try** 块内随时都有可能发生异常, [OpenWrite\(\)](#) 调用本身也有可能失败, 所以我们无法保证该文件在我们试图关闭它时处于打开状态。**finally** 块添加了一项检查, 以确保在调用 [Close](#) 方法前, [FileStream](#) 对象不为 **null**。如果没有 **null** 检查, **finally** 块可能引发自身的 [NullReferenceException](#), 但是应当尽可能避免在 **finally** 块中引发异常。

在 **finally** 块中关闭数据库连接是另一个不错的选择。因为有时候数据库服务器允许的连接数是有限的, 所以尽快关闭数据库连接很重要。在由于引发了异常而无法关闭连接的情况下, 使用 **finally** 块也是比等待垃圾回收更好的一种选择。

请参见

参考

[异常和异常处理 \(C# 编程指南\)](#)

[异常处理 \(C# 编程指南\)](#)

[using 语句 \(C# 参考\)](#)

[try-catch \(C# 参考\)](#)

[try-finally \(C# 参考\)](#)

[try-catch-finally\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

如何：捕捉非 CLS 异常

包括 C++/CLI 在内的某些 .NET 语言允许对象引发不是由 [Exception](#) 派生的异常。这类异常称为“非 CLS 异常”或“非异常”。在 Visual C# 中，您不能引发非 CLS 异常，但是可以采用两种方法捕捉这类异常：

- 在 `catch (Exception e)` 块中作为 [RuntimeWrappedException](#) 捕捉。

默认情况下，Visual C# 程序集将非 CLS 异常作为包装异常来捕捉。如果需要访问可通过 [WrappedException](#) 属性访问的原始异常，请使用此方法。下面的过程解释如何使用此方法捕捉异常。

- 在位于 `catch (Exception)` 或 `catch (Exception e)` 块之后的常规 catch 块（没有指定异常类型的 catch 块）中。

如果您要执行某个操作（如写入日志文件）以响应非 CLS 异常，并且不需要访问异常信息，请使用此方法。默认情况下，公共语言运行库会包装所有异常。要禁用此行为，请将程序集级别属性 `[assembly:`

`RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]` 添加到您的代码中，该属性通常在 `AssemblyInfo.cs` 文件中。

捕捉非 CLS 异常

1. 在 `catch (Exception e)` block 中，使用 **as** 关键字来测试 `e` 能否强制转换为 [RuntimeWrappedException](#)。
2. 通过 **WrappedException** 属性访问原始异常。

示例

下面的示例演示如何捕捉由使用 C++/CLR 编写的类库引发的非 CLS 异常。请注意，在此示例中，Visual C# 客户端代码事先已经知道要引发的异常类型为 [System.String](#)。只要 **WrappedException** 属性的原始类型可通过您的代码来访问，就可以将该属性强制转换回其原始类型。

```
// Class library written in C++/CLR
ThrowNonCLS.Class1 myClass = new ThrowNonCLS.Class1();

try
{
    // throws gcnew System::String(
    // "I do not derive from System.Exception!");
    myClass.TestThrow();
}

catch (Exception e)
{
    RuntimeWrappedException rwe = e as RuntimeWrappedException;
    if (rwe != null)
    {
        String s = rwe.WrappedException as String;
        if (s != null)
        {
            Console.WriteLine(s);
        }
    }
    else
    {
        // handle other System.Exception types
    }
}
```

请参见

参考

[异常和异常处理\(C# 编程指南\)](#)

[RuntimeWrappedException](#)

互操作性(C# 编程指南)

互操作性使您能够保留和利用在现有非托管代码中的投入。运行在公共语言运行库 (CLR) 的控制之下的代码称为“托管代码”，运行在 CLR 之外的代码称为“非托管代码”。COM、COM+、C++ 组件、ActiveX 组件和 Win32 API 都是非托管代码的示例。

.NET Framework 通过平台调用服务、[System.Runtime.InteropServices](#) 命名空间和 CLR 以及通过 COM 互操作性 (COM interop) 支持与非托管代码的互操作性。

- 存在两种从托管代码使用非托管 API 的方法：通过平台调用服务和通过 C++ 中的 It Just Works (IJW)。平台调用使托管代码能够调用从非托管动态链接库 (DLL) 导出的函数，比如 Win32 API 和自定义 DLL。CLR 处理 DLL 加载和参数封送处理。出于性能的考虑，应考虑 .NET Framework 中是否有可用的等价函数而不是使用平台调用。有关更多信息，请参见[平台调用详解](#)。
- COM interop，它使托管代码能够通过 COM 接口与 COM 客户端与 COM 对象交互。存在两种从托管代码使用 COM 组件的方法：
 - 对于调用与 OLE 自动化兼容的 COM 组件，可使用 COM interop 或 tlbimp.exe。CLR 将处理 COM 组件激活和参数封送处理。
 - 对于基于 IDL 的 COM 组件，可使用 IJW 和 C++。实现 [IUnknown](#)、[IDispatch](#) 和其他标准 COM 接口的每个公共托管类都可以通过 COM Interop 从非托管代码调用。有关更多信息，请参见["Microsoft .NET/COM Migration and Interoperability"](#)(Microsoft .NET/COM 迁移和互操作性)。

有关更多信息，请参见[与非托管代码交互操作](#)和[Improving Interop Performance\(改进互操作的性能\)](#)。

`PInvoke` 和 COM interop 都使用封送处理在托管和非托管代码之间转换诸如整数、字符串、数组、结构和指针等参数。有关更多信息，请参见[Interop 封送处理概述](#)。

相关章节

[互操作性概述\(C# 编程指南\)](#)

[如何：使用 COM Interop 创建 Excel 电子表格\(C# 编程指南\)](#)

[如何：使用平台调用播放波形文件\(C# 编程指南\)](#)

[如何：使用 COM Interop 进行 Word 拼写检查\(C# 编程指南\)](#)

[COM 类示例\(C# 编程指南\)](#)

[如何：将托管代码用作 Excel 的自动化外接程序\(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 10.5.7 外部方法
- 17.5 互操作的属性
- 18.8 动态内存分配

请参见

概念

[C# 编程指南](#)

其他资源

[.NET Compact Framework 中的互操作性](#)

互操作性概述 (C# 编程指南)

用于在 C# 托管代码和非托管代码之间进行互操作的工具和技术包括平台调用服务以及 .NET Framework 和 COM 互操作性工具。

平台调用

平台调用将非托管代码作为导出函数进行定位和调用。它还根据需要封送调用的参数，如输入和输出参数、整型、字符串、数组和结构等。有关更多信息，请参见[用平台调用封送数据和使用非托管 DLL 函数](#)。

注意

公共语言运行库 (CLR) 管理对系统资源的访问。在 CLR 外调用非托管代码会避开此安全机制，因此会造成安全风险。例如，非托管代码可能避开 CLR 安全机制，直接调用非托管代码中的资源。有关更多信息，请参见[“.NET Framework Security”\(.NET Framework 安全性\)](#)。

.NET Framework 和 COM 之间的互操作性工具

- 若要从托管代码中调用 COM API，请使用[类型库导入程序 \(Tlbimp.exe\)](#)，它以类型库作为输入，并输出 .NET Framework 程序集和关联托管元数据。然后，可以将此生成的 .NET Framework 程序集作为项目引用添加到 Visual Studio 项目中。例如，使用 `TlbImp comlibrary.dll /out: comlibrary.dll` 并在项目中添加对 `comlibrary.dll` 的引用。有关更多信息，请参见[将类型库当作程序集导入](#)和位于[“Calling COM Components from .NET Clients”\(从 .NET 客户端调用 COM 组件\)](#)。
- 若要从 COM 调用托管代码，请使用[类型库导出程序 \(Tlbexp.exe\)](#)，它以托管的程序集作为输入，生成一个类型库，其中包含该程序集所定义的全部 **public** 类型的 COM 定义。
- 若要从 COM 客户端调用托管组件，请使用[程序集注册工具 \(Regasm.exe\)](#)，它读取 .NET Framework 程序集中的元数据并添加注册表项，以便 COM 客户端创建托管类。
- 若要调用 ActiveX 控件，请使用[Windows 窗体 ActiveX 控件导入程序 \(Aximp.exe\)](#)，它以 ActiveX 控件的类型库作为输入，生成的包装控件允许控件寄宿在 Windows 窗体中。例如，`Aximp activex.ocx` 创建两个文件：`viz.activex.dll` 和 `Axactivex.dll`。当项目引用自动生成的 `Axactivex.dll` 时，可以使用这两个文件。

注意

若要使用上述工具（如为 **COMinterop** 注册托管程序集），您必须具有管理员或超级用户安全权限。有关更多信息，请参见[“.NET Framework Security”\(.NET Framework 安全性\)](#)。

互操作性示例和技术

有关更多信息，请参见[Interop 封送处理概述](#)。有关互操作性技术的更多信息，请参见：

- [如何：使用 PInvoke 播放波形文件](#)
- [如何：将托管代码用作 Excel 的自动化外接程序](#)
- [如何：使用 COM interop 在 Word 中进行拼写检查](#)
- [如何：使用 COM Interop 创建 Excel 电子表格](#)

请参见

概念

[C# 编程指南](#)

[自定义封送处理概述](#)

其他资源

[用 COM Interop 对数据进行封送处理](#)

[.NET Compact Framework 中的互操作性](#)

[互操作封送处理](#)

如何：使用 COM Interop 创建 Excel 电子表格 (C# 编程指南)

下面的代码示例演示如何使用 **COM interop** 创建 **Excel** 电子表格。有关 **Excel** 的更多信息，请参见“[Microsoft Excel Objects](#)”(Microsoft Excel 对象)和“[Open Method](#)”(Open 方法)

此示例演示如何在 C# 中使用 .NET Framework **COM interop** 功能打开现有的 **Excel** 电子表格。使用 **Excel** 程序集来打开 **Excel** 电子表格中一定范围内的单元格并向其中输入数据。

注意

为使此代码正常运行，必须在系统中安装 **Excel**。

注意

显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

使用 COM interop 创建 Excel 电子表格

1. 在 Visual Studio 中创建新的 C# 控制台应用程序，将其命名为 **CreateExcelWorksheet**。
2. 将 **Excel** 程序集作为引用添加到项目中：右键单击项目，选择“添加引用”。
3. 单击“添加引用”对话框的“COM”选项卡，找到“Microsoft Excel 11 对象库”。
4. 双击“Microsoft Excel 11 对象库”，然后按“确定”。

注意

根据安装的 Office 版本的不同，Excel 程序集可能名为“Excel 10 对象库”或“Excel 11 对象库”。

5. 复制下面的代码并将其粘贴到 `Program.cs` 文件中覆盖原有内容。

C#

```

using System;
using System.Reflection;
using Microsoft.Office.Interop.Excel;

public class CreateExcelWorksheet
{
    static void Main()
    {
        Microsoft.Office.Interop.Excel.Application xlApp = new Microsoft.Office.Interop.Excel.Application();

        if (xlApp == null)
        {
            Console.WriteLine("EXCEL could not be started. Check that your office installation and project references are correct.");
            return;
        }
        xlApp.Visible = true;

        Workbook wb = xlApp.Workbooks.Add(XlWBATemplate.xlWBATWorksheet);
        Worksheet ws = (Worksheet)wb.Worksheets[1];

        if (ws == null)
        {
            Console.WriteLine("Worksheet could not be created. Check that your office

```

```
installation and project references are correct.");
}

// Select the Excel cells, in the range c1 to c7 in the worksheet.
Range aRange = ws.get_Range("C1", "C7");

if (aRange == null)
{
    Console.WriteLine("Could not get a range. Check to be sure you have the correct versions of the office DLLs.");
}

// Fill the cells in the C1 to C7 range of the worksheet with the number 6.
Object[] args = new Object[1];
args[0] = 6;
aRange.GetType().InvokeMember("Value", BindingFlags SetProperty, null, aRange,
args);

// Change the cells in the C1 to C7 range of the worksheet to the number 8.
aRange.Value2 = 8;
}
}
```

安全

若要使用 **COM interop**, 您必须拥有“管理员”或“超级用户”安全权限。有关安全的更多信息, 请参见[“.NET Framework Security”\(.NET Framework 安全性\)](#)。

请参见

任务

[如何: 使用 COM Interop 进行 Word 拼写检查\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[互操作性概述\(C# 编程指南\)](#)

其他资源

[.NET Compact Framework 中的互操作性](#)

[COM 互操作性示例](#)

如何：使用平台调用播放波形文件 (C# 编程指南)

下面的 C# 代码示例演示如何在 Windows 平台上使用平台调用服务来播放波形声音文件。

示例

此代码示例使用 **DllImport** 将 **winmm.dll** 的 **PlaySound** 方法入口点作为 **Form1 PlaySound()** 导入。该示例具有一个带有一个按钮的简单 Windows 窗体。单击该按钮可打开标准 Windows **OpenFileDialog** 对话框，以便打开要播放的文件。选择波形文件后，将使用 **winmm.DLL** 程序集方法的 **PlaySound()** 方法来播放此文件。有关 **winmm.dll** 的 **PlaySound** 方法的更多信息，请参见 [Using PlaySound to Play Waveform-Audio Files \(使用 PlaySound 播放波形音频文件\)](#)。浏览并选择一个带 .wav 扩展名的文件，然后单击“打开”来使用平台调用播放该波形文件。将在文本框中显示选择的文件的完整路径。

“打开文件”对话框会进行筛选，以便只显示具有 .wav 扩展名的文件，其筛选器设置为：

C#

```
dialog1.Filter = "Wav Files (*.wav)|*.wav";
```

C#

```
using System.Windows.Forms;

namespace WinSound
{
    public partial class Form1 : Form
    {
        private TextBox textBox1;
        private Button button1;

        public Form1() //constructor
        {
            InitializeComponent();
        }

        [System.Runtime.InteropServices.DllImport("winmm.DLL", EntryPoint = "PlaySound", SetLastError = true)]
        private static extern bool PlaySound(string szSound, System.IntPtr hMod, PlaySoundFlags flags);

        [System.Flags]
        public enum PlaySoundFlags : int
        {
            SND_SYNC = 0x0000,
            SND_ASYNC = 0x0001,
            SND_NODEFAULT = 0x0002,
            SND_LOOP = 0x0008,
            SND_NOSTOP = 0x0010,
            SND_NOWAIT = 0x00002000,
            SND_FILENAME = 0x00020000,
            SND_RESOURCE = 0x00040004
        }

        private void button1_Click (object sender, System.EventArgs e)
        {
            OpenFileDialog dialog1 = new OpenFileDialog();

            dialog1.Title = "Browse to find sound file to play";
            dialog1.InitialDirectory = @"c:\";
            dialog1.Filter = "Wav Files (*.wav)|*.wav";
            dialog1.FilterIndex = 2;
            dialog1.RestoreDirectory = true;

            if(dialog1.ShowDialog() == DialogResult.OK)
            {
                textBox1.Text = dialog1.FileName;
            }
        }
    }
}
```

```
        PlaySound (dialog1.FileName, new System.IntPtr(), PlaySoundFlags.SND_SYNC);
    }
}
}
```

编译代码

编译代码

1. 在 Visual Studio 中创建一个新的 C# Windows 应用程序项目，并命名为 **WinSound**。
2. 复制上面的代码并将其粘贴到 `Form1.cs` 文件中以覆盖原来的内容。
3. 复制下面的代码，并将它粘贴到 `Form1.Designer.cs` 文件的 `InitializeComponent()` 方法中现有代码的后面。

C#

```
this.button1 = new System.Windows.Forms.Button();
this.textBox1 = new System.Windows.Forms.TextBox();
this.SuspendLayout();
//
// button1
//
this.button1.Location = new System.Drawing.Point(192, 40);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(88, 24);
this.button1.TabIndex = 0;
this.button1.Text = "Browse";
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(8, 40);
this.textBox1.Name = "textBox1";
this.textBox1.Size = new System.Drawing.Size(168, 20);
this.textBox1.TabIndex = 1;
this.textBox1.Text = "FILE path";
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(5, 13);
this.ClientSize = new System.Drawing.Size(292, 266);
this.Controls.Add(this.textBox1);
this.Controls.Add(this.button1);
this.Name = "Form1";
this.Text = "Platform Invoke WinSound C#";
this.ResumeLayout(false);
this.PerformLayout();
```

4. 编译并运行代码。

安全

有关更多信息，请参见“[.NET Framework Security](#)”(.NET Framework 安全性)。

请参见

任务

[平台调用技术示例](#)

[概念](#)

[C# 编程指南](#)

[互操作性概述 \(C# 编程指南\)](#)

[其他资源](#)

[用平台调用封送数据](#)

如何：使用 COM Interop 进行 Word 拼写检查 (C# 编程指南)

下面的代码示例演示如何在 Visual C# 应用程序中通过 COM 互操作性使用 Word 的拼写检查功能。有关更多信息，请参见“[ProofreadingErrors Collection Object](#)”(ProofreadingErrors 集合对象)和“[Microsoft Word Objects](#)”(Microsoft Word 对象)。

示例

此示例演示如何从 C# 应用程序使用 Word 的拼写检查器。它使用 COM 互操作性创建一个新的 **Word.application** 对象。然后使用 **Range** 对象上的 **ProofreadingErrors** 集合并查找该范围中有拼写错误的单词。

C#

```

using System.Reflection;
using Word = Microsoft.Office.Interop.Word;

namespace WordSpell
{
    public partial class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TextBox textBox1;
        private System.Windows.Forms.Button button1;
        private System.Windows.Forms.Label label1;

        public Form1() //constructor
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, System.EventArgs e)
        {
            Word.Application app = new Word.Application();

            int errors = 0;
            if (textBox1.Text.Length > 0)
            {
                app.Visible = false;

                // Setting these variables is comparable to passing null to the function.
                // This is necessary because the C# null cannot be passed by reference.
                object template = Missing.Value;
                object newTemplate = Missing.Value;
                object documentType = Missing.Value;
                object visible = true;

                Word._Document doc1 = app.Documents.Add(ref template, ref newTemplate, ref
documentType, ref visible);
                doc1.Words.First.InsertBefore(textBox1.Text);
                Word.ProofreadingErrors spellErrorsColl = doc1.SpellingErrors;
                errors = spellErrorsColl.Count;

                object optional = Missing.Value;

                doc1.CheckSpelling(
                    ref optional, ref optional, ref optional, ref optional, ref optional, r
ef optional,
                    ref optional, ref optional, ref optional, ref optional, ref optional, r
ef optional);

                label1.Text = errors + " errors corrected ";
                object first = 0;
                object last = doc1.Characters.Count - 1;
                textBox1.Text = doc1.Range(ref first, ref last).Text;
            }

            object saveChanges = false;
            object originalFormat = Missing.Value;
            object routeDocument = Missing.Value;
        }
    }
}

```

```
        app.Quit(ref saveChanges, ref originalFormat, ref routeDocument);
    }
}
```

编译代码

此示例要求系统上安装有 Word，并根据已安装的 Office 版本的不同，Word 程序集可能名为“Microsoft office 10 对象库”或“Word 11 对象库”。

注意

显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

编译代码

1. 在 Visual Studio 中创建一个新的 C# Windows 应用程序项目，并将它命名为 **WordSpell**。
2. 复制上面的代码并将其粘贴到 `Form1.cs` 文件中以覆盖原来的内容。
3. 复制下面的代码，并将它粘贴到 `Form1.Designer.cs` 文件的 `InitializeComponent()` 方法中现有代码的后面。

C#

```
this.textBox1 = new System.Windows.Forms.TextBox();
this.button1 = new System.Windows.Forms.Button();
this.label1 = new System.Windows.Forms.Label();
this.SuspendLayout();
//
// textBox1
//
this.textBox1.Location = new System.Drawing.Point(40, 40);
this.textBox1.Multiline = true;
this.textBox1.Name = "textBox1";
this.textBox1.ScrollBars = System.Windows.Forms.ScrollBars.Vertical;
this.textBox1.Size = new System.Drawing.Size(344, 136);
this.textBox1.TabIndex = 0;
this.textBox1.Text = "";
//
// button1
//
this.button1.Location = new System.Drawing.Point(392, 40);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(96, 23);
this.button1.TabIndex = 1;
this.button1.Text = "Check Spelling";
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// label1
//
this.label1.Location = new System.Drawing.Point(40, 24);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(336, 16);
this.label1.TabIndex = 2;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(5, 13);
this.ClientSize = new System.Drawing.Size(496, 205);
```

```
this.Controls.Add(this.label1);
this.Controls.Add(this.button1);
this.Controls.Add(this.textBox1);
this.Name = "Form1";
this.Text = "SpellCheckDemo";
this.ResumeLayout(false);
```

4. 将 **Word** 程序集作为引用添加到该项目中。右击该项目，单击“添加引用”，再单击“添加引用”对话框的“COM”选项卡。双击“Microsoft Office 11 对象库”，然后按“确定”。

注意，显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 Visual Studio 设置。

安全

若要使用 COM 互操作性，您必须具有管理员或超级用户安全权限。有关更多信息，请参见[“.NET Framework Security”\(.NET Framework 安全性\)](#)。

请参见

任务

[如何：使用 COM Interop 创建 Excel 电子表格\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[互操作性概述\(C# 编程指南\)](#)

其他资源

[COM 互操作性示例](#)

COM 类示例 (C# 编程指南)

下面是一个公开为 COM 对象的类的示例。将这段代码放置在 .cs 文件中并添加到项目后，请将“为 COM Interop 注册”属性设置为“True”。有关更多信息，请参见[如何：为 COM Interop 注册组件](#)。

向 COM 公开 Visual C# 对象要求声明一个类接口、一个事件接口（如果需要）和类本身。类成员必须遵循下列规则才能对 COM 可见：

- 类必须是公共的。
- 属性、方法和事件必须是公共的。
- 属性和方法必须在类接口上声明。
- 事件必须在事件接口中声明。

其他没有在这些接口中声明的类的公共成员对于 COM 是不可见的，但它们对于其他 .NET Framework 对象将是可见的。

若要向 COM 公开属性 (Property) 和方法，必须在类接口上声明这些属性 (Property) 和方法，并用 **DispId** 属性 (Attribute) 予以标记，然后在类中实现它们。成员在接口中声明的顺序即是用于 COM vtable 的顺序。

若要从类中公开事件，必须在事件接口上声明这些事件，并用 **DispId** 属性予以标记。该类不应实现此接口。

类实现类接口；它可以实现多个接口，但第一个实现将作为默认类接口。在此处实现向 COM 公开的方法和属性。它们必须标记为是公共的，并且必须与类接口中的声明匹配。同时，在此处声明由类引发的事件。它们必须标记为是公共的，并且必须与事件接口中的声明匹配。

有关更多信息，请参见[“COM Interop 第一部分”示例](#)、[“COM Interop 第二部分”示例](#)和[COM 互操作性示例](#)。

示例

C#

```
using System.Runtime.InteropServices;

namespace project_name
{
    [Guid("EAA4976A-45C3-4BC5-BC0B-E474F4C3C83F")]
    public interface ComClass1Interface
    {
    }

    [Guid("7BD20046-DF8C-44A6-8F6B-687FAA26FA71"),
     InterfaceType(ComInterfaceType.InterfaceIsIDispatch)]
    public interface ComClass1Events
    {
    }

    [Guid("0D53A3E8-E51A-49C7-944E-E72A2064F938"),
     ClassInterface(ClassInterfaceType.None),
     ComSourceInterfaces(typeof(ComClass1Events))]
    public class ComClass1 : ComClass1Interface
    {
    }
}
```

[请参见](#)

[参考](#)

[互操作性 \(C# 编程指南\)](#)

[“项目设计器”->“生成”页 \(C#\)](#)

[概念](#)

[C# 编程指南](#)

如何：将托管代码用作 Excel 的自动化外接程序 (C# 编程指南)

通过 Excel 的自动化外接程序，可以将 COM 库的公共函数作为单元格公式进行调用。下面的示例演示如何创建 C# 外接程序，用于在 Excel 工作表的单元格中计算所得税税率。[ComRegisterFunctionAttribute](#) 自动注册该外接程序，无需任何其他工具即可将托管代码注册为 COM 程序集。有关更多信息，请参见[互操作性概述 \(C# 编程指南\)](#)。

注意

显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见[Visual Studio 设置](#)。

计算所得税

给定个人年收入，即可使用典型税表计算所得税。例如，下面为一个假设的个人所得税表。

示例税表

- 如果收入在零美元和 \$7,000 之间，则税款为金额的 10%。
- 如果收入在 \$7,000 和 \$28,400 之间，则税款是超过 \$7,000 的部分乘以 15% 再加上 \$700.00。
- 如果收入在 \$28,400 和 \$68,800 之间，则税款是超过 \$28,400 的部分乘以 25% 再加上 3,910.00。
- 如果收入在 \$68,800 和 \$143,500 之间，则税款是超过 \$68,800 的部分乘以 28% 再加上 \$14,010.00。
- 如果收入在 \$143,500 和 \$311,950 之间，则税款是超过 \$143,500 的部分乘以 33% 再加上 \$34,926.00。
- 如果收入高于 \$311,950，则税款是超过 \$311,950 的部分乘以 35% 再加上 \$90,514.50。

使用 Visual Studio 和托管代码创建 Excel 的自动化外接程序

- 创建名为 **ExcelAddIn** 的新 Visual C#“类库”项目。
- 在“项目”菜单中，单击“属性”，然后在“生成”窗格中单击。单击标有“为 COM Interop 注册”的复选框。此设置会自动针对 COM 互操作性将您的程序集注册。
- 将下面的代码粘贴到类文件中。

C#

```
using System.Runtime.InteropServices;

namespace TaxTables
{
    [ClassInterface(ClassInterfaceType.AutoDual)]
    public class TaxTables
    {
        public static double Tax(double income)
        {
            if (income > 0 && income <= 7000) {return (.10 * income);}
            if (income > 7000 && income <= 28400) {return 700.00 + (.15 * (income - 7000));}
            if (income > 28400 && income <= 68800) {return 3910.00 + (.25 * (income - 28400));}
            if (income > 68800 && income <= 143500) {return 14010.00 + (.28 * (income - 68800));}
            if (income > 143500 && income <= 311950) {return 34926.00 + (.33 * (income - 143500));}
            if (income > 311950) {return 90514.50 + (.35 * (income - 311950));}
            return 0;
    }
}
```

```
}

[ComRegisterFunctionAttribute]
public static void RegisterFunction(System.Type t)
{
    Microsoft.Win32.Registry.ClassesRoot.CreateSubKey
        ("CLSID\\\" + t.GUID.ToString().ToUpper() + "}\\Programmable");
}

[ComUnregisterFunctionAttribute]
public static void UnregisterFunction(System.Type t)
{
    Microsoft.Win32.Registry.ClassesRoot.DeleteSubKey
        ("CLSID\\\" + t.GUID.ToString().ToUpper() + "}\\Programmable");
}
}
```

运行代码

运行 Excel 外接程序

- 生成 **ExcelAddIn** 项目，按 F5 进行编译。
- 在 Excel 中打开一个新工作簿。
- 从“工具”菜单上单击“外接程序”，然后单击“自动化”。
- 在“自动化服务器”对话框中，选择外接程序列表中的“ExcelAddIn”，然后单击“确定”。
- 在一个工作簿单元格中，键入 =Tax(23500)。该单元格将显示 3175。
- 若要在将 ExcelAddIn.dll 移动到其他目录后进行注册，请运行带 **/codebase** 的 **regasm**。可能会收到警告，指示该程序集未签名。

安全

若要使用 COM 互操作性，您必须具有管理员或超级用户的安全权限。有关更多信息，请参见[“.NET Framework Security”\(.NET Framework 安全性\)](#)。

请参见

任务

[如何：使用 COM Interop 创建 Excel 电子表格\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[互操作性概述\(C# 编程指南\)](#)

线程处理(C# 编程指南)

线程处理使 C# 程序能够执行并发处理，以便您可以同时执行多个操作。例如，您可以使用线程处理来监视用户输入，执行后台任务，以及处理并发输入流。[System.Threading](#) 命名空间提供支持多线程编程的类和接口，使您可以轻松地执行创建和启动新线程，同步多个线程，挂起线程以及中止线程等任务。

若要在 C# 代码中合并线程处理，只需创建一个将在主线程外执行的函数，并让一个新的 [Thread](#) 对象指向该函数即可。下面的代码示例在 C# 应用程序中创建一个新线程：

C#

```
System.Threading.Thread newThread;  
newThread = new System.Threading.Thread(anObject.AMethod);
```

下面的代码示例在 C# 应用程序中启动一个新线程：

C#

```
newThread.Start();
```

多线程处理可解决响应性和多任务的问题，但同时引入了资源共享和同步问题，因为根据中央线程调度机制，线程将在没有警告的情况下中断和继续。有关更多信息，请参见[线程同步](#)。有关概述信息，请参见[使用线程和线程处理](#)。

概述

线程具有以下特点：

- 线程使 C# 程序能够执行并发处理。
- .NET Framework 的 **System.Threading** 命名空间使线程更易于使用。
- 线程共享应用程序的资源。有关更多信息，请参见[使用线程和线程处理](#)。

相关章节

有关更多信息，请参见下列主题：

- [使用线程处理](#)
- [如何：创建和终止线程](#)
- [如何：使用线程池](#)
- [如何：对制造者线程和使用者线程进行同步](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.10 执行顺序
- 8.12 lock 语句
- 10.4.3 可变字段
- 10.7.1 类似字段的事件

请参见

任务

[监视器同步技术示例](#)

[等待同步技术示例](#)

参考

[Mutex](#)

[概念](#)

[C# 编程指南](#)

[Mutex](#)

[监视器](#)

[互锁操作](#)

[AutoResetEvent](#)

[委托 \(C# 编程指南\)](#)

[其他资源](#)

[Thread 类](#)

[如何:通过使用 Visual C# .NET 在多线程处理环境中同步对共享资源的访问](#)

使用线程处理(C# 编程指南)

默认情况下，C# 程序具有一个线程。此线程执行程序中以 `Main` 方法开始和结束的代码。`Main` 直接或间接执行的每一个命令都由默认线程(或主线程)执行，当 `Main` 返回时此线程也将终止。不过，可以创建辅助线程，以便与主线程一起并行执行代码。这些线程通常称为“辅助线程”。

辅助线程可以用于执行耗时的任务或时间要求紧迫的任务，而不必占用主线程。例如，辅助线程通常用在服务器应用程序中，以便不必等待前面的请求完成即可响应传入的请求。辅助线程还可用于在桌面应用程序中执行“后台”任务，以便使主线程(用于驱动用户界面元素)保持对用户操作的响应。

多线程处理解决了吞吐量和响应性的问题，但同时也带来了资源共享问题，如死锁和争用状态。多线程特别适用于需要不同资源(如文件句柄和网络连接)的任务。为单个资源分配多个线程可能会导致同步问题，线程会被频繁阻止以等待其他线程，从而与使用多线程的初衷背道而驰。

常见的策略是使用辅助线程执行不需要大量占用其他线程所使用的资源的耗时任务或时间要求紧迫的任务。实际上，程序中的某些资源必须由多个线程访问。考虑到这些情况，`System.Threading` 命名空间提供了用于同步线程的类。这些类包括 `Mutex`、`Monitor`、`Interlocked`、`AutoResetEvent` 和 `ManualResetEvent`。

您可以使用这些类中的部分或所有类来同步多个线程的活动，但是某些多线程处理支持由 C# 语言支持。例如，C# 中的 `Lock` 语句通过隐式使用 `Monitor` 来提供同步功能。

下面的主题演示了常用的多线程处理技术：

- [如何: 创建线程、启动线程和在线程之间交互](#)
- [如何: 对制造者线程和使用者线程进行同步](#)
- [如何: 使用线程池](#)

相关章节

有关更多信息，请参见：

- [如何: 使用 Visual C# .NET 创建线程](#)
- [如何: 使用 Visual C# .NET 向线程池提交工作项](#)
- [如何: 通过使用 Visual C# .NET 在多线程处理环境中同步对共享资源的访问](#)
- [线程与线程处理](#)
- [为多线程处理同步数据](#)

请参见

任务

[监视器同步技术示例](#)

[等待同步技术示例](#)

[“线程”示例](#)

参考

[Mutex](#)

概念

[C# 编程指南](#)

监视器

[互锁操作](#)

[AutoResetEvent](#)

其他资源

[托管线程处理](#)

[使用线程和线程处理](#)

[线程示例](#)

[Thread 类](#)

线程同步 (C# 编程指南)

以下各节描述了在多线程应用程序中可以用来同步资源访问的功能和类。

在应用程序中使用多个线程的一个好处是每个线程都可以异步执行。对于 Windows 应用程序，耗时的任务可以在后台执行，而使应用程序窗口和控件保持响应。对于服务器应用程序，多线程处理提供了用不同线程处理每个传入请求的能力。否则，在完全满足前一个请求之前，将无法处理每个新请求。

然而，线程的异步特性意味着必须协调对资源（如文件句柄、网络连接和内存）的访问。否则，两个或更多的线程可能在同一时间访问相同的资源，而每个线程都不知道其他线程的操作。结果将产生不可预知的数据损坏。

对于整数数据类型的简单操作，可以用 [Interlocked](#) 类的成员来实现线程同步。对于其他所有数据类型和非线程安全的资源，只有使用本主题中的结构才能安全地执行多线程处理。

有关多线程编程的背景信息，请参见：

- [使用线程处理 \(C# 编程指南\)](#)
- [托管线程处理基本知识](#)
- [使用线程和线程处理](#)
- [托管线程处理的最佳做法](#)

lock 关键字

lock 关键字可以用来确保代码块完成运行，而不会被其他线程中断。这是通过在代码块运行期间为给定对象获取互斥锁来实现的。

lock 语句以关键字 **lock** 开头，它有一个作为参数的对象，在该参数的后面还有一个一次只能由一个线程执行的代码块。例如：

C#

```
public void Function()
{
    System.Object lockThis = new System.Object();
    lock(lockThis)
    {
        // Access thread-sensitive resources.
    }
}
```

提供给 **lock** 关键字的参数必须为基于引用类型的对象，该对象用来定义锁的范围。在上例中，锁的范围限定为此函数，因为函数外不存在任何对该对象的引用。严格地说，提供给 **lock** 的对象只是用来唯一地标识由多个线程共享的资源，所以它可以是任意类实例。然而，实际上，此对象通常表示需要进行线程同步的资源。例如，如果一个容器对象将被多个线程使用，则可以将该容器传递给 **lock**，而 **lock** 后面的同步代码块将访问该容器。只要其他线程在访问该容器前先锁定该容器，则对该对象的访问将是安全同步的。

通常，最好避免锁定 **public** 类型或锁定不受应用程序控制的对象实例。例如，如果该实例可以被公开访问，则 **lock(this)** 可能会有问题，因为不受控制的代码也可能会锁定该对象。这可能导致死锁，即两个或更多个线程等待释放同一对象。出于同样的原因，锁定公共数据类型（相比于对象）也可能导致问题。锁定字符串尤其危险，因为字符串被公共语言运行库（CLR）“暂留”。这意味着整个程序中任何给定字符串都只有一个实例，就是这同一个对象表示了所有运行的应用程序域的所有线程中的该文本。因此，只要在应用程序进程中的任何位置处具有相同内容的字符串上放置了锁，就将锁定应用程序中该字符串的所有实例。因此，最好锁定不会被暂留的私有或受保护成员。某些类提供专门用于锁定的成员。例如，[Array](#) 类型提供 [SyncRoot](#)。许多集合类型也提供 [SyncRoot](#)。

有关 **lock** 关键字的更多信息，请参见：

- [lock 语句 \(C# 参考\)](#)
- [如何：对制造者线程和使用者线程进行同步 \(C# 编程指南\)](#)

监视器

与 **lock** 关键字类似，监视器防止多个线程同时执行代码块。[Enter](#) 方法允许一个且仅一个线程继续执行后面的语句；其他所有线程都将被阻止，直到执行语句的线程调用 [Exit](#)。这与使用 **lock** 关键字一样。事实上，**lock** 关键字就是用 [Monitor](#) 类来实现

的。例如：

C#

```
lock(x)
{
    DoSomething();
}
```

这等效于：

C#

```
System.Object obj = (System.Object)x;
System.Threading.Monitor.Enter(obj);
try
{
    DoSomething();
}
finally
{
    System.Threading.Monitor.Exit(obj);
}
```

使用 **lock** 关键字通常比直接使用 **Monitor** 类更可取，一方面是因为 **lock** 更简洁，另一方面是因为 **lock** 确保了即使受保护的代码引发异常，也可以释放基础监视器。这是通过 **finally** 关键字来实现的，无论是否引发异常它都执行关联的代码块。

有关监视器的更多信息，请参见[监视器同步技术示例](#)。

同步事件和等待句柄

使用锁或监视器对于防止同时执行区分线程的代码块很有用，但是这些构造不允许一个线程向另一个线程传达事件。这需要“同步事件”，它是有两个状态（终止和非终止）的对象，可以用来激活和挂起线程。让线程等待非终止的同步事件可以将线程挂起，将事件状态更改为终止可以将线程激活。如果线程试图等待已经终止的事件，则线程将继续执行，而不会延迟。

同步事件有两种：[AutoResetEvent](#) 和 [ManualResetEvent](#)。它们之间唯一的不同在于，无论何时，只要 [AutoResetEvent](#) 激活线程，它的状态将自动从终止变为非终止。相反，[ManualResetEvent](#) 允许它的终止状态激活任意多个线程，只有当它的 [Reset](#) 方法被调用时才还原到非终止状态。

可以通过调用一种等待方法，如 [WaitOne](#)、[WaitAny](#) 或 [WaitAll](#)，让线程等待事件。[System.Threading.WaitHandle.WaitOne](#) 使线程一直等待，直到单个事件变为终止状态；[System.Threading.WaitHandle.WaitAny](#) 阻止线程，直到一个或多个指示的事件变为终止状态；[System.Threading.WaitHandle.WaitAll](#) 阻止线程，直到所有指示的事件都变为终止状态。当调用事件的 [Set](#) 方法时，事件将变为终止状态。

在下面的示例中，创建了一个线程，并由 [Main](#) 函数启动该线程。新线程使用 [WaitOne](#) 方法等待一个事件。在该事件被执行 [Main](#) 函数的主线程终止之前，该线程一直处于挂起状态。一旦该事件终止，辅助线程将返回。在本示例中，因为事件只用于一个线程的激活，所以使用 [AutoResetEvent](#) 或 [ManualResetEvent](#) 类都可以。

C#

```
using System;
using System.Threading;

class ThreadingExample
{
    static AutoResetEvent autoEvent;

    static void DoWork()
    {
        Console.WriteLine("    worker thread started, now waiting on event...");
        autoEvent.WaitOne();
        Console.WriteLine("    worker thread reactivated, now exiting...");
    }

    static void Main()
    {
        autoEvent = new AutoResetEvent(false);
```

```

        Console.WriteLine("main thread starting worker thread...");
        Thread t = new Thread(DoWork);
        t.Start();

        Console.WriteLine("main thrad sleeping for 1 second...");
        Thread.Sleep(1000);

        Console.WriteLine("main thread signaling worker thread...");
        autoEvent.Set();
    }
}

```

有关线程同步事件用法的更多示例, 请参见:

- [监视器同步技术示例](#)
- [读取器—编写器同步技术示例](#)
- [线程池技术示例](#)
- [等待同步技术示例](#)

Mutex 对象

mutex 与监视器类似; 它防止多个线程在同一时间同时执行某个代码块。事实上, 名称“mutex”是术语“互相排斥 (mutually exclusive)”的简写形式。然而与监视器不同的是, mutex 可以用来使跨进程的线程同步。mutex 由 [Mutex](#) 类表示。

当用于进程间同步时, mutex 称为“命名 mutex”, 因为它将用于另一个应用程序, 因此它不能通过全局变量或静态变量共享。必须给它指定一个名称, 才能使两个应用程序访问同一个 mutex 对象。

尽管 mutex 可以用于进程内的线程同步, 但是使用 [Monitor](#) 通常更为可取, 因为监视器是专门为 .NET Framework 而设计的, 因而它可以更好地利用资源。相比之下, [Mutex](#) 类是 Win32 构造的包装。尽管 mutex 比监视器更为强大, 但是相对于 [Monitor](#) 类, 它所需要的互操作转换更消耗计算资源。有关 mutex 的用法示例, 请参见 [Mutex](#)。

相关章节

- [如何: 创建和终止线程 \(C# 编程指南\)](#)
- [如何: 使用线程池 \(C# 编程指南\)](#)
- [如何: 通过使用 Visual C# .NET 在多线程处理环境中同步对共享资源的访问](#)
- [如何: 使用 Visual C# .NET 创建线程](#)
- [如何: 使用 Visual C# .NET 向线程池提交工作项](#)
- [如何: 通过使用 Visual C# .NET 在多线程处理环境中同步对共享资源的访问](#)

请参见

参考

[Thread](#)

[WaitOne](#)

[WaitAny](#)

[WaitAll](#)

[Monitor](#)

[Mutex](#)

[AutoResetEvent](#)

[ManualResetEvent](#)

[Interlocked](#)

[WaitHandle](#)

概念

[C# 编程指南](#)

如何：创建和终止线程 (C# 编程指南)

此示例演示如何创建辅助线程，并用它与主线程并行执行处理。还将演示如何使一个线程等待另一个线程，并正确地终止线程。有关多线程处理的背景信息，请参见[托管线程处理](#)和[使用线程处理 \(C# 编程指南\)](#)。

该示例创建一个名为 `Worker` 的类，该类包含辅助线程将执行的方法 `DoWork`。这实际上是辅助线程的 `Main` 函数。辅助线程将通过调用此方法来开始执行，并在此方法返回时自动终止。`DoWork` 方法如下所示：

C#

```
public void DoWork()
{
    while (!_shouldStop)
    {
        Console.WriteLine("worker thread: working...");
    }
    Console.WriteLine("worker thread: terminating gracefully.");
}
```

`Worker` 类包含另一个方法，该方法用于通知 `DoWork` 它应当返回。此方法名为 `RequestStop`，如下所示：

C#

```
public void RequestStop()
{
    _shouldStop = true;
}
```

`RequestStop` 方法只是将 `true` 赋给 `_shouldStop` 数据成员。由于此数据成员由 `DoWork` 方法来检查，因此这会间接导致 `DoWork` 返回，从而终止辅助线程。但是，需要注意：`DoWork` 和 `RequestStop` 将由不同线程执行。`DoWork` 由辅助线程执行，而 `RequestStop` 由主线程执行，因此 `_shouldStop` 数据成员声明为 **volatile**，如下所示：

C#

```
private volatile bool _shouldStop;
```

volatile 关键字用于通知编译器，将有多个线程访问 `_shouldStop` 数据成员，因此它不应当对此成员的状态做任何优化假设。有关更多信息，请参见[volatile \(C# 参考\)](#)。

通过将 **volatile** 与 `_shouldStop` 数据成员一起使用，可以从多个线程安全地访问此成员，而不需要使用正式的线程同步技术，但这仅仅是因为 `_shouldStop` 是 **bool**。这意味着只需要执行单个原子操作就能修改 `_shouldStop`。但是，如果此数据成员是类、结构或数组，那么，从多个线程访问它可能会导致间歇的数据损坏。假设有一个更改数组中的值的线程。Windows 定期中断线程，以便允许其他线程执行，因此线程会在分配某些数组元素之后和分配其他元素之前被中断。这意味着，数组现在有了一个程序员从不想要的状态，因此，读取此数组的另一个线程可能会失败。

在实际创建辅助线程之前，`Main` 函数会创建一个 `Worker` 对象和 `Thread` 的一个实例。线程对象被配置为：通过将对 `Worker.DoWork` 方法的引用传递给 `Thread` 构造函数，来将该方法用作入口点，如下所示：

C#

```
Worker workerObject = new Worker();
Thread workerThread = new Thread(workerObject.DoWork);
```

此时，尽管辅助线程对象已存在并已配置，但尚未创建实际的辅助线程。只有当 `Main` 调用 `Start` 方法后，才会创建实际的辅助线程：

C#

```
workerThread.Start();
```

此时，系统将启动辅助线程的执行，但这是在与主线程异步执行的。这意味着 Main 函数将在辅助线程进行初始化的同时继续执行代码。为了保证 Main 函数不会尝试在辅助线程有机会执行之前将它终止，Main 函数将一直循环，直到辅助线程对象的 IsAlive 属性设置为 true：

C#

```
while (!workerThread.IsAlive);
```

下一步，通过调用 Sleep 来将主线程中断片刻。这保证了辅助线程的 DoWork 函数在 Main 函数执行其他任何命令之前，在 DoWork 方法内部执行若干次循环：

C#

```
Thread.Sleep(1);
```

在 1 毫秒之后，Main 将通知辅助线程对象，它应当使用 Worker.RequestStop 方法（前面已介绍）自行终止：

C#

```
workerObject.RequestStop();
```

还可以通过调用 Abort 来从一个线程终止另一个线程，但这会强行终止受影响的线程，而不管它是否已完成自己的任务，并且不提供清理资源的机会。此示例中显示的技术是首选方法。

最后，Main 函数对辅助线程对象调用 Join 方法。此方法导致当前线程阻塞或等待，直到对象所表示的线程终止。因此，直到辅助线程返回后，Join 才会返回，然后自行终止：

C#

```
workerThread.Join();
```

此时，只有执行 Main 的主线程还存在。它会显示一条最终消息，然后返回，从而使主线程也终止。

下面显示了完整的示例：

示例

C#

```
using System;
using System.Threading;

public class Worker
{
    // This method will be called when the thread is started.
    public void DoWork()
    {
        while (!_shouldStop)
        {
            Console.WriteLine("worker thread: working...");
        }
        Console.WriteLine("worker thread: terminating gracefully.");
    }
    public void RequestStop()
    {
        _shouldStop = true;
    }
    // Volatile is used as hint to the compiler that this data
    // member will be accessed by multiple threads.
    private volatile bool _shouldStop;
}

public class WorkerThreadExample
{
    static void Main()
```

```

{
    // Create the thread object. This does not start the thread.
    Worker workerObject = new Worker();
    Thread workerThread = new Thread(workerObject.DoWork);

    // Start the worker thread.
    workerThread.Start();
    Console.WriteLine("main thread: Starting worker thread...");

    // Loop until worker thread activates.
    while (!workerThread.IsAlive);

    // Put the main thread to sleep for 1 millisecond to
    // allow the worker thread to do some work:
    Thread.Sleep(1);

    // Request that the worker thread stop itself:
    workerObject.RequestStop();

    // Use the Join method to block the current thread
    // until the object's thread terminates.
    workerThread.Join();
    Console.WriteLine("main thread: Worker thread has terminated.");
}
}

```

示例输出

```

main thread: starting worker thread...
worker thread: working...
worker thread: terminating gracefully...
main thread: worker thread has terminated

```

请参见

[任务](#)

[“线程”示例](#)

[参考](#)

[线程处理\(C# 编程指南\)](#)

[使用线程处理\(C# 编程指南\)](#)

[volatile\(C# 参考\)](#)

[Thread](#)

[Mutex](#)

[Monitor](#)

[Start](#)

[IsAlive](#)

[Sleep](#)

[Join](#)

[Abort](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[托管线程处理](#)

[线程示例](#)

如何：对制造者线程和使用者线程进行同步 (C# 编程指南)

下面的示例演示使用 **lock** 关键字以及 [AutoResetEvent](#) 和 [ManualResetEvent](#) 类对主线程和两个辅助线程进行线程同步。有关更多信息，请参见 [lock 语句 \(C# 参考\)](#)。

该示例创建两个辅助线程。一个线程生成元素并将它们存储在非线程安全的泛型队列中。有关更多信息，请参见 [Queue](#)。另一个线程使用此队列中的项。另外，主线程定期显示队列的内容，因此该队列被三个线程访问。**lock** 关键字用于同步对队列的访问，以确保队列的状态没有被破坏。

除了用 **lock** 关键字来阻止同时访问外，还用两个事件对象提供进一步的同步。一个事件对象用来通知辅助线程终止，另一个事件对象由制造者线程用来在有新项添加到队列中时通知使用者线程。这两个事件对象封装在一个名为 [SyncEvents](#) 的类中。这使事件可以轻松传递到表示制造者线程和使用者线程的对象。[SyncEvents](#) 类是按如下方式定义的：

C#

```
public class SyncEvents
{
    public SyncEvents()
    {

        _newItemEvent = new AutoResetEvent(false);
        _exitThreadEvent = new ManualResetEvent(false);
        _eventArray = new WaitHandle[2];
        _eventArray[0] = _ newItemEvent;
        _eventArray[1] = _ exitThreadEvent;
    }

    public EventWaitHandle ExitThreadEvent
    {
        get { return _exitThreadEvent; }
    }
    public EventWaitHandle NewItemEvent
    {
        get { return _ newItemEvent; }
    }
    public WaitHandle[] EventArray
    {
        get { return _eventArray; }
    }

    private EventWaitHandle _ newItemEvent;
    private EventWaitHandle _ exitThreadEvent;
    private WaitHandle[] _ eventArray;
}
```

[AutoResetEvent](#) 类用于“新项”事件，因为您希望每当使用者线程响应此事件后，此事件都能自动重置。或者，将 [ManualResetEvent](#) 类用于“退出”事件，因为您希望当此事件终止时有多个线程响应。如果您改为使用 [AutoResetEvent](#)，则仅在一个线程响应该事件以后，该事件就还原到非终止状态。另一个线程不会响应，因此在这种情况下，将无法终止。

[SyncEvents](#) 类创建两个事件，并将它们以两种不同的形式存储：一种是作为 [EventWaitHandle](#)（它是 [AutoResetEvent](#) 和 [ManualResetEvent](#) 的基类），一种是作为基于 [WaitHandle](#) 的数组。如关于使用者线程的讨论中所述，此数组是必需的，因为它使使用者线程可以响应两个事件中的任何一个。

使用者线程和制造者线程分别由名为 [Consumer](#) 和 [Producer](#) 的类表示，这两个类都定义名为 [ThreadRun](#) 的方法。这些方法用作 [Main](#) 方法创建的辅助线程的入口点。

[Producer](#) 类定义的 [ThreadRun](#) 方法如下所示：

C#

```
// Producer.ThreadRun
public void ThreadRun()
{
    int count = 0;
    Random r = new Random();
```

```

    while (!_syncEvents.ExitThreadEvent.WaitOne(0, false))
    {
        lock (((ICollection)_queue).SyncRoot)
        {
            while (_queue.Count < 20)
            {
                _queue.Enqueue(r.Next(0, 100));
                _syncEvents.NewItemEvent.Set();
                count++;
            }
        }
    }
    Console.WriteLine("Producer thread: produced {0} items", count);
}

```

此方法一直循环，直到“退出线程”事件变为终止状态。此事件的状态由 `WaitOne` 方法使用 `SyncEvents` 类定义的 `ExitThreadEvent` 属性测试。在这种情况下，检查该事件的状态不会阻止当前线程，因为 `WaitOne` 使用的第一个参数为零，这表示该方法应立即返回。如果 `WaitOne` 返回 `true`，则说明该事件当前处于终止状态。如果是这样，`ThreadRun` 方法将返回，其效果相当于终止执行此方法的辅助线程。

在“退出线程”事件终止前，`Producer.ThreadStart` 方法将尝试在队列中保持 20 个项。每个项是 0 到 100 之间的一个整数。在添加新项前，该集合必须处于锁定状态，以防止使用者线程和主线程同时访问该集合。这是使用 `lock` 关键字来实现的。传递给 `lock` 的参数是通过 `ICollection` 接口公开的 `SyncRoot` 字段。此字段专门为同步线程访问而提供。对集合的独占访问权限被授予 `lock` 后面的代码块中包含的所有指令。对于制造者添加到队列中的每个新项，都将调用“新项”事件的 `Set` 方法。这将通知使用者线程离开挂起状态并开始处理新项。

`Consumer` 对象还定义名为 `ThreadRun` 的方法。与制造者的 `ThreadRun` 类似，此方法由 `Main` 方法创建的辅助线程执行。然而，使用者的 `ThreadStart` 必须响应两个事件。`Consumer.ThreadRun` 方法如下所示：

C#

```

// Consumer.ThreadRun
public void ThreadRun()
{
    int count = 0;
    while (WaitHandle.WaitAny(_syncEvents.EventArray) != 1)
    {
        lock (((ICollection)_queue).SyncRoot)
        {
            int item = _queue.Dequeue();
        }
        count++;
    }
    Console.WriteLine("Consumer Thread: consumed {0} items", count);
}

```

此方法使用 `WaitAny` 来阻止使用者线程，直到所提供的数组中的任意一个等待句柄变为终止状态。在这种情况下，数组中有两个句柄，一个用来终止辅助线程，另一个用来指示有新项添加到集合中。`WaitAny` 返回变为终止状态的事件的索引。“新项”事件是数组中的第一个事件，因此索引为零表示新项。在这种情况下，检查索引为 1 的项（它表示“退出线程”事件），以确定此方法是否继续使用项。如果“新项”事件处于终止状态，您将通过 `lock` 获得对集合的独占访问权限并使用新项。因为此示例生成并使用数千个项，所以不显示使用的每个项，而是使用 `Main` 定期显示队列中的内容，如下面所演示的那样。

`Main` 方法从创建队列（队列中的内容将被生成和使用）和 `SyncEvents` 的实例（已在前面演示）开始：

C#

```

Queue<int> queue = new Queue<int>();
SyncEvents syncEvents = new SyncEvents();

```

然后，`Main` 配置 `Producer` 和 `Consumer` 对象以供辅助线程使用。然而，此步骤并不创建或启动实际的辅助线程：

C#

```

Producer producer = new Producer(queue, syncEvents);

```

```
Consumer consumer = new Consumer(queue, syncEvents);
Thread producerThread = new Thread(producer.ThreadRun);
Thread consumerThread = new Thread(consumer.ThreadRun);
```

请注意，队列和同步事件对象作为构造函数参数同时传递给 `Consumer` 和 `Producer` 线程。这为两个对象提供了它们执行各自任务所需的共享资源。然后创建两个新的 `Thread` 对象，并对每个对象使用 `ThreadRun` 方法作为参数。每个辅助线程在启动时都将此参数用作线程的入口点。

然后，`Main` 通过调用 `Start` 方法来启动两个辅助线程，如下所示：

C#

```
producerThread.Start();
consumerThread.Start();
```

此时，创建了两个新的辅助线程，它们独立于当前正在执行 `Main` 方法的主线程开始异步执行过程。事实上，`Main` 接下来要做的事情是通过调用 `Sleep` 方法将主线程挂起。该方法将当前正在执行的线程挂起指定的时间(毫秒)。在此时间间隔过后，`Main` 将重新激活，这时它将显示队列的内容。`Main` 重复此过程四次，如下所示：

C#

```
for (int i=0; i<4; i++)
{
    Thread.Sleep(2500);
    ShowQueueContents(queue);
}
```

最后，`Main` 通过调用“退出线程”事件的 `Set` 方法通知辅助线程终止，然后对每个辅助线程调用 `Join` 方法以阻止主线程，直到每个辅助线程都响应该事件并终止。

有一个线程同步的最终示例：`ShowQueueContents` 方法。与制造者线程和使用者线程类似，此方法使用 `lock` 获得对队列的独占访问权限。然而在这种情况下，独占访问尤其重要，因为 `ShowQueueContents` 对整个集合进行枚举。对集合进行枚举是一个很容易由于异步操作造成数据损坏的操作，因为它涉及遍历整个集合的内容。`ShowQueueContents` 方法如下所示：

C#

```
syncEvents.ExitThreadEvent.Set();

producerThread.Join();
consumerThread.Join();
```

最后请注意，`ShowQueueContents` 是由主线程执行的，因为它被 `Main` 调用。这意味着当此方法获得对队列的独占访问权限时，它实际上既阻止了制造者线程访问队列，也阻止了使用者线程访问队列。`ShowQueueContents` 锁定队列并枚举其内容：

C#

```
private static void ShowQueueContents(Queue<int> q)
{
    lock (((ICollection)q).SyncRoot)
    {
        foreach (int item in q)
        {
            Console.Write("{0} ", item);
        }
    }
    Console.WriteLine();
}
```

下面是完整的示例。

示例

C#

```

using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;

public class SyncEvents
{
    public SyncEvents()
    {

        _ newItemEvent = new AutoResetEvent(false);
        _ exitThreadEvent = new ManualResetEvent(false);
        _ eventArray = new WaitHandle[2];
        _ eventArray[0] = _ newItemEvent;
        _ eventArray[1] = _ exitThreadEvent;
    }

    public EventWaitHandle ExitThreadEvent
    {
        get { return _ exitThreadEvent; }
    }
    public EventWaitHandle NewItemEvent
    {
        get { return _ newItemEvent; }
    }
    public WaitHandle[] EventArray
    {
        get { return _ eventArray; }
    }

    private EventWaitHandle _ newItemEvent;
    private EventWaitHandle _ exitThreadEvent;
    private WaitHandle[] _ eventArray;
}

public class Producer
{
    public Producer(Queue<int> q, SyncEvents e)
    {
        _ queue = q;
        _ syncEvents = e;
    }
    // Producer.ThreadRun
    public void ThreadRun()
    {
        int count = 0;
        Random r = new Random();
        while (!_ syncEvents.ExitThreadEvent.WaitOne(0, false))
        {
            lock (((ICollection)_ queue).SyncRoot)
            {
                while (_ queue.Count < 20)
                {
                    _ queue.Enqueue(r.Next(0,100));
                    _ syncEvents.NewItemEvent.Set();
                    count++;
                }
            }
        }
        Console.WriteLine("Producer thread: produced {0} items", count);
    }
    private Queue<int> _ queue;
    private SyncEvents _ syncEvents;
}

public class Consumer
{
    public Consumer(Queue<int> q, SyncEvents e)
    {
}

```

```

        _queue = q;
        _syncEvents = e;
    }
    // Consumer.ThreadRun
    public void ThreadRun()
    {
        int count = 0;
        while (WaitHandle.WaitAny(_syncEvents.EventArray) != 1)
        {
            lock (((ICollection)_queue).SyncRoot)
            {
                int item = _queue.Dequeue();
            }
            count++;
        }
        Console.WriteLine("Consumer Thread: consumed {0} items", count);
    }
    private Queue<int> _queue;
    private SyncEvents _syncEvents;
}

public class ThreadSyncSample
{
    private static void ShowQueueContents(Queue<int> q)
    {
        lock (((ICollection)q).SyncRoot)
        {
            foreach (int item in q)
            {
                Console.Write("{0} ", item);
            }
        }
        Console.WriteLine();
    }

    static void Main()
    {
        Queue<int> queue = new Queue<int>();
        SyncEvents syncEvents = new SyncEvents();

        Console.WriteLine("Configuring worker threads...");
        Producer producer = new Producer(queue, syncEvents);
        Consumer consumer = new Consumer(queue, syncEvents);
        Thread producerThread = new Thread(producer.ThreadRun);
        Thread consumerThread = new Thread(consumer.ThreadRun);

        Console.WriteLine("Launching producer and consumer threads...");
        producerThread.Start();
        consumerThread.Start();

        for (int i=0; i<4; i++)
        {
            Thread.Sleep(2500);
            ShowQueueContents(queue);
        }

        Console.WriteLine("Signaling threads to terminate...");
        syncEvents.ExitThreadEvent.Set();

        producerThread.Join();
        consumerThread.Join();
    }
}

```

示例输出

```
Configuring worker threads...
Launching producer and consumer threads...
22 92 64 70 13 59 9 2 43 52 91 98 50 96 46 22 40 94 24 87
79 54 5 39 21 29 77 77 1 68 69 81 4 75 43 70 87 72 59
0 69 98 54 92 16 84 61 30 45 50 17 86 16 59 20 73 43 21
38 46 84 59 11 87 77 5 53 65 7 16 66 26 79 74 26 37 56 92
Signalling threads to terminate...
Consumer Thread: consumed 1053771 items
Producer thread: produced 1053791 items
```

请参见

任务

[监视器同步技术示例](#)

[等待同步技术示例](#)

参考

[线程同步 \(C# 编程指南\)](#)

[lock 语句 \(C# 参考\)](#)

[AutoResetEvent](#)

[ManualResetEvent](#)

[Set](#)

[Join](#)

[WaitOne](#)

[WaitAll](#)

[Queue](#)

[ICollection](#)

[Start](#)

[Sleep](#)

[WaitHandle](#)

[EventWaitHandle](#)

概念

[C# 编程指南](#)

[其他资源](#)

[Thread 类](#)

如何：使用线程池 (C# 编程指南)

"线程池"是可以用来在后台执行多个任务的线程集合。(有关背景信息, 请参见[使用线程处理](#)。)这使主线程可以自由地异步执行其他任务。

线程池通常用于服务器应用程序。每个传入请求都将分配给线程池中的一个线程, 因此可以异步处理请求, 而不会占用主线程, 也不会延迟后续请求的处理。

一旦池中的某个线程完成任务, 它将返回到等待线程队列中, 等待被再次使用。这种重用使应用程序可以避免为每个任务创建新线程的开销。

线程池通常具有最大线程数限制。如果所有线程都繁忙, 则额外的任务将放入队列中, 直到有线程可用时才能够得到处理。

您可以实现自己的线程池, 但是通过 [ThreadPool](#) 类使用 .NET Framework 提供的线程池更容易一些。

下面的示例使用 .NET Framework 线程池计算 20 和 40 之间的十个数的 Fibonacci 结果。每个 Fibonacci 结果都由 Fibonacci 类表示, 该类提供一种名为 [ThreadPoolCallback](#) 的方法来执行此计算。将创建表示每个 Fibonacci 值的对象, [ThreadPoolCallback](#) 方法将传递给 [QueueUserWorkItem](#), 它分配池中的一个可用线程来执行此方法。

由于为每个 Fibonacci 对象都提供了一个半随机值来进行计算, 而且十个线程都将竞争处理器时间, 因此无法提前知道十个结果全部计算出来所需的时间。因此在构造期间为每个 Fibonacci 对象传递 [ManualResetEvent](#) 类的一个实例。当计算完成时, 每个对象都通知提供的事件对象, 使主线程用 [WaitAll](#) 阻止执行, 直到十个 Fibonacci 对象全部计算出了结果。然后 [Main](#) 方法将显示每个 Fibonacci 结果。

示例

C#

```
using System;
using System.Threading;

public class Fibonacci
{
    public Fibonacci(int n, ManualResetEvent doneEvent)
    {
        _n = n;
        _doneEvent = doneEvent;
    }

    // Wrapper method for use with thread pool.
    public void ThreadPoolCallback(Object threadContext)
    {
        int threadIndex = (int)threadContext;
        Console.WriteLine("thread {0} started...", threadIndex);
        _fibOfN = Calculate(_n);
        Console.WriteLine("thread {0} result calculated...", threadIndex);
        _doneEvent.Set();
    }

    // Recursive method that calculates the Nth Fibonacci number.
    public int Calculate(int n)
    {
        if (n <= 1)
        {
            return n;
        }

        return Calculate(n - 1) + Calculate(n - 2);
    }

    public int N { get { return _n; } }
    private int _n;

    public int FibOfN { get { return _fibOfN; } }
    private int _fibOfN;

    private ManualResetEvent _doneEvent;
```

```

}

public class ThreadPoolExample
{
    static void Main()
    {
        const int FibonacciCalculations = 10;

        // One event is used for each Fibonacci object
        ManualResetEvent[] doneEvents = new ManualResetEvent[FibonacciCalculations];
        Fibonacci[] fibArray = new Fibonacci[FibonacciCalculations];
        Random r = new Random();

        // Configure and launch threads using ThreadPool:
        Console.WriteLine("launching {0} tasks...", FibonacciCalculations);
        for (int i = 0; i < FibonacciCalculations; i++)
        {
            doneEvents[i] = new ManualResetEvent(false);
            Fibonacci f = new Fibonacci(r.Next(20,40), doneEvents[i]);
            fibArray[i] = f;
            ThreadPool.QueueUserWorkItem(f.ThreadPoolCallback, i);
        }

        // Wait for all threads in pool to calculation...
        WaitHandle.WaitAll(doneEvents);
        Console.WriteLine("All calculations are complete.");

        // Display the results...
        for (int i= 0; i<FibonacciCalculations; i++)
        {
            Fibonacci f = fibArray[i];
            Console.WriteLine("Fibonacci({0}) = {1}", f.N, f.FibOfN);
        }
    }
}

```

示例输出

```

launching 10 tasks...
result calculated...
all calculations complete
Fibonacci(22) = 17711
Fibonacci(25) = 75025
Fibonacci(32) = 2178309
Fibonacci(36) = 14930352
Fibonacci(32) = 2178309
Fibonacci(26) = 121393
Fibonacci(35) = 9227465
Fibonacci(23) = 28657
Fibonacci(39) = 63245986
Fibonacci(22) = 17711

```

请参见

任务

[监视器同步技术示例](#)

[等待同步技术示例](#)

参考

[线程处理\(C# 编程指南\)](#)

[使用线程处理\(C# 编程指南\)](#)

[Mutex](#)

[WaitAll](#)

[ManualResetEvent](#)

[Set](#)

[ThreadPool](#)

[QueueUserWorkItem](#)

[ManualResetEvent](#)

[概念](#)

[C# 编程指南](#)

[监视器](#)

[其他资源](#)

[.NET Framework 中的安全性](#)

[如何:通过使用 Visual C# .NET 在多线程处理环境中同步对共享资源的访问](#)

性能(C# 编程指南)

此部分讨论可能对性能造成负面影响的两个问题，并提供指向介绍性能问题的相关资源的链接。

装箱和取消装箱

装箱和取消装箱都是需要大量运算的过程。对值类型进行装箱时，必须创建一个全新的对象。此操作所需时间可比赋值操作长 20 倍。取消装箱时，强制转换过程所需时间可达赋值操作的四倍。有关更多信息，请参见[装箱和取消装箱](#)。

析构函数

不应使用空析构函数。如果类包含析构函数，Finalize 队列中则会创建一个项。调用析构函数时，将调用垃圾回收器来处理该队列。如果析构函数为空，只会导致性能降低。有关更多信息，请参见[析构函数](#)。

其他资源

- [Writing Faster Managed Code: Know What Things Cost](#)(编写更快的托管代码:了解开销)
- [Writing High-Performance Managed Applications : A Primer](#)(编写高性能的托管应用程序:入门)
- [Garbage Collector Basics and Performance Hints](#)(垃圾回收器基础和性能提示)
- [Performance Tips and Tricks in .NET Applications](#)(.NET 应用程序的性能提示和技巧)

请参见

参考

[安全性\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

反射 (C# 编程指南)

反射提供了封装程序集、模块和类型的对象 ([Type](#) 类型)。可以使用反射动态创建类型的实例，将类型绑定到现有对象，或从现有对象获取类型并调用其方法或访问其字段和属性。如果代码中使用了属性，可以利用反射对它们进行访问。有关更多信息，请参见[属性](#)。

下面是使用静态方法 **GetType**-- 从 **Object** 基类派生的所有类型都继承该方法 -- 获取变量类型的简单反射示例：

C#

```
// Using GetType to obtain type information:
int i = 42;
System.Type type = i.GetType();
System.Console.WriteLine(type);
```

输出为：

System.Int32

此示例使用反射获取已加载的程序集的完整名称：

C#

```
// Using Reflection to get information from an Assembly:
System.Reflection.Assembly o = System.Reflection.Assembly.Load("mscorlib.dll");
System.Console.WriteLine(o.GetName());
```

输出为：

mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

反射概述

反射在下列情况下很有用：

- 需要访问程序元数据的属性。请参见主题[使用反射访问属性](#)。
- 检查和实例化程序集中的类型。
- 在运行时构建新类型。使用 [System.Reflection.Emit](#) 中的类。
- 执行后期绑定，访问在运行时创建的类型的方法。请参见主题[动态加载和使用类型](#)。

相关章节

更多信息：

- [反射概述](#)
- [查看类型信息](#)
- [反射类型和泛型类型](#)
- [System.Reflection.Emit](#)
- [使用反射访问属性 \(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.12 属性
- 7.5.11 `typeof` 运算符

请参见 参考

[应用程序域 \(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[程序集和全局程序集缓存 \(C# 编程指南\)](#)

C# DLL(C# 编程指南)

您可以使用 Visual C# 来创建可由其他 .NET 应用程序和非托管代码调用的 DLL。

本节内容

[如何: 创建和使用 C# DLL\(C# 编程指南\)](#)

显示如何在 Visual C# 中创建类库项目。

相关章节

[/baseaddress\(指定 DLL 的基址\)\(C# 编译器选项\)](#)

[创建用于容纳 DLL 函数的类](#)

[为什么 DLL 断点不起作用？](#)

[调试 DLL 项目](#)

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

如何：创建和使用 C# DLL(C# 编程指南)

动态链接库 (DLL) 在运行时链接到程序。为说明如何生成和使用 DLL, 请看以下方案:

- `MathLibrary.dll`: 为库文件, 其中包含运行时要调用的方法。此例中, DLL 包含两个方法:`Add` 和 `Multiply`。
- `Add.cs`: 为源文件, 其中包含 `Add(long i, long j)` 方法。该方法返回参数之和。包含 `Add` 方法的 `AddClass` 类是命名空间 `UtilityMethods` 的成员。
- `Mult.cs`: 为源文件, 其中包含 `Multiply(long x, long y)` 方法。该方法返回参数之积。包含 `Multiply` 方法的 `MultiplyClass` 类也是命名空间 `UtilityMethods` 的成员。
- `TestCode.cs`: 包含 `Main` 方法的文件。它使用 DLL 文件中的方法来计算运行时参数的和与积。

示例

C#

```
// File: Add.cs
namespace UtilityMethods
{
    public class AddClass
    {
        public static long Add(long i, long j)
        {
            return (i + j);
        }
    }
}
```

C#

```
// File: Mult.cs
namespace UtilityMethods
{
    public class MultiplyClass
    {
        public static long Multiply(long x, long y)
        {
            return (x * y);
        }
    }
}
```

C#

```
// File: TestCode.cs

using UtilityMethods;

class TestCode
{
    static void Main(string[] args)
    {
        System.Console.WriteLine("Calling methods from MathLibrary.dll:");

        if (args.Length != 2)
        {
            System.Console.WriteLine("Usage: TestCode <num1> <num2>");
            return;
        }

        long num1 = long.Parse(args[0]);
        long num2 = long.Parse(args[1]);
```

```
        long sum = AddClass.Add(num1, num2);
        long product = MultiplyClass.Multiply(num1, num2);

        System.Console.WriteLine("{0} + {1} = {2}", num1, num2, sum);
        System.Console.WriteLine("{0} * {1} = {2}", num1, num2, product);
    }
}
```

此文件包含使用 DLL 方法 Add 和 Multiply 的算法。它首先分析从命令行输入的参数 num1 和 num2。然后使用 AddClass 类中的 Add 方法计算和，使用 MultiplyClass 类中的 Multiply 方法计算积。

请注意，文件开头的 **using** 指令使您得以在编译时使用未限定的类名来引用 DLL 方法，如下所示：

C#

```
MultiplyClass.Multiply(num1, num2);
```

否则，必须使用完全限定名，如下所示：

C#

```
UtilityMethods.MultiplyClass.Multiply(num1, num2);
```

执行

若要运行程序，请输入 EXE 文件的名称，文件名的后面跟两个数字，如下所示：

```
TestCode 1234 5678
```

输出

```
Calling methods from MathLibrary.DLL:
1234 + 5678 = 6912
1234 * 5678 = 7006652
```

编译代码

若要生成文件 MathLibrary.DLL，请使用以下命令行编译文件 Add.cs 和文件 Mult.cs：

```
csc /target:library /out:MathLibrary.DLL Add.cs Mult.cs
```

/target:library 编译器选项通知编译器输出 DLL 文件而不是 EXE 文件。后跟文件名的 **/out** 编译器选项用于指定 DLL 文件名。否则，编译器使用第一个文件 (Add.cs) 作为 DLL 文件名。

若要生成可执行文件 TestCode.exe，请使用以下命令行：

```
csc /out:TestCode.exe /reference:MathLibrary.DLL TestCode.cs
```

/out 编译器选项通知编译器输出 EXE 文件并且指定输出文件名 (TestCode.exe)。此编译器选项是可选的。**/引用** 编译器选项指定该程序使用的 DLL 文件。

请参见

任务

[如何：为 DLL 指定基址](#)

概念

[C# 编程指南](#)

[创建用于容纳 DLL 函数的类](#)

安全性(C# 编程指南)

安全是每个 C# 应用程序的一个非常重要的方面，在每个开发阶段都必须考虑：而不仅仅是在完成设计和实现后才需要考虑。

特定于 C# 的安全建议

本列表并未列出所有潜在的安全问题。它强调 C# 开发人员需要知道的一些常见问题。

- 使用 [checked](#) 关键字控制整型算术运算和转换的溢出检查上下文。
- 始终对参数使用最严格的数据类型。例如，在将一个值传入描述数据结构大小的方法时，应使用无符号整数而不是整数。
- 不要根据文件名作出决定。文件名可以用多种不同的方式表示，因而检测是否有特定文件时可能会跳过该文件。
- 千万不要将密码或其他敏感信息硬编码到应用程序中。
- 始终验证用于生成 SQL 查询的输入。
- 验证传入方法的所有输入。[System.Text.RegularExpressions](#) 命名空间中的正则表达式方法对于确认输入（如电子邮件地址）的格式是否正确很有用。
- 不要显示异常信息：它会给任何潜在的攻击者提供有价值的线索。
- 确保应用程序在最低的可能特权下运行时能够正常工作。少数应用程序要求用户作为管理员登录。
- 不要使用自己的加密算法，应使用 [System.Security.Cryptography](#) 类。
- 为程序集指定强名称。
- 不要在 XML 或其他配置文件中存储敏感信息。
- 仔细检查包装本机代码的托管代码。确保本机代码是安全的，尤其是在防止缓冲区溢出方面。
- 在使用从应用程序之外传入的委托时应保持谨慎。
- 对程序集运行 [FxCop](#) 以确保符合 Microsoft .NET Framework 设计准则。FxCop 还可以查找 200 多种代码缺陷并针对这些代码缺陷发出警告。

本节内容

以下 MSDN 主题详细介绍了如何创建安全、可靠的软件。

- [Microsoft Security Developer Center \(Microsoft 安全开发人员中心\)](#).
- [Defend Your Apps and Critical User Info with Defensive Coding Techniques \(使用防御性编码技术保护应用程序和关键用户信息\)](#).
- [An Overview of Security in the .NET Framework \(.NET Framework 安全概述\)](#).
- [Defend Your Code with Top Ten Security Tips Every Developer Must Know \(使用每个开发人员都必须知道的十大安全技巧来保护代码\)](#).
- [Security Brief: Beware of Fully Trusted Code \(安全简报：小心完全受信任的代码\)](#).
- [Programmatically check for canonicalization issues with ASP.NET \(以编程方式检查 ASP.NET 的规范化问题\)](#).
- [Windows 窗体中的安全性概述](#)
- [编写安全托管控件](#)

请参见

概念

[C# 编程指南](#)

C# 参考

这部分提供有关 C# 关键字、运算符以及编译器错误和警告的参考资料。

本节内容

[C# 关键字](#)

C# 关键字和语法。

[C# 运算符](#)

C# 运算符和语法。

[C# 预处理器指令](#)

用于在 C# 源代码中嵌入的编译器命令。

[C# 编译器选项](#)

编译器选项及其用法。

[C# 术语](#)

C# 词和短语的术语表。

相关章节

[C# 语言规范](#)

提供指向最新版本的 C# 语言规范(Microsoft Word 格式)的指针。

[C# 常见问题](#)

在 C# Developer Center 中提供内容不断增加的 C# 常见问题列表。

[Microsoft 知识库中的 C# 知识库文章](#)

动态搜索存储在 MSDN 上的 C# 相关知识库文章。

[Visual C#](#)

提供 Visual C# 文档门户。

[Visual C# 示例](#)

提供 Visual C# 的列表及链接。

[Visual C# 代码编辑器功能](#)

提供一些链接，这些链接指向描述 IDE 和编辑器的概念性主题及任务主题。

[用 Visual C# 编写应用程序](#)

提供一些链接，这些链接指向说明如何执行某些常见编程任务的主题。

请参见

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C#](#)

C# 关键字

关键字是对编译器具有特殊意义的预定义保留标识符。它们不能在程序中用作标识符，除非它们有一个 @ 前缀。例如，@if 是一个合法的标识符，而 if 不是合法的标识符，因为它是关键字。

abstract	event	new	struct
as	explicit	null	switch
base	extern	object	this
bool	false	operator	throw
break	finally	out	true
byte	fixed	override	try
case	float	params	typeof
catch	for	private	uint
char	foreach	protected	ulong
checked	goto	public	unchecked
class	if	readonly	unsafe
const	implicit	ref	ushort
continue	in	return	using
decimal	int	sbyte	virtual
default	interface	sealed	volatile
delegate	internal	short	void
do	is	sizeof	while
double	lock	stackalloc	
else	long	static	
enum	namespace	string	

上下文关键字

get	partial	set
value	where	yield

请参见

概念

[C# 编程指南](#)

[其他资源](#)

类型(C# 参考)

C# 类型体系包含下列几种类别：

- [值类型](#)
- [引用类型](#)
- [指针类型](#)

值类型的变量存储数据，而引用类型的变量存储对实际数据的引用。引用类型也称为对象。指针类型仅可用于 `unsafe` 模式。

通过[装箱和取消装箱](#)，可以将值类型转换为引用类型，然后再转换回值类型。除了装箱值类型外，无法将引用类型转换为值类型。

本节还介绍 `void` 类型。

值类型也可以为空，这意味着它们能存储加法非值状态。有关更多信息，请参见[可空类型](#)。

请参见

参考

[C# 关键字](#)

[强制转换\(C# 编程指南\)](#)

[数据类型\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[类型参考表\(C# 参考\)](#)

值类型(C# 参考)

值类型主要由两类组成：

- [结构](#)
- [枚举](#)

结构分为以下几类：

- Numeric(数值)类型
 - [整型](#)
 - [浮点型](#)
 - [decimal](#)
- [bool](#)
- 用户定义的结构。

值类型的主要功能

基于值类型的变量直接包含值。将一个值类型变量赋给另一个值类型变量时，将复制包含的值。这与引用类型变量的赋值不同，引用类型变量的赋值只复制对对象的引用，而不复制对象本身。

所有的值类型均隐式派生自 [System.ValueType](#)。

与引用类型不同，从值类型不可能派生出新的类型。但与引用类型相同的是，结构也可以实现接口。

与引用类型不同，值类型不可能包含 **null** 值。然而，[可空类型](#)功能允许将 **null** 赋给值类型。

每种植类型均有一个隐式的默认构造函数来初始化该类型的默认值。有关值类型默认值的信息，请参见[默认值表](#)。

简单类型的主要功能

所有的简单类型(C# 语言的组成部分)均为 .NET Framework 系统类型的别名。例如，[int](#) 是 [System.Int32](#) 的别名。有关完整的别名列表，请参见[内置类型表\(C# 参考\)](#)。

编译时计算操作数均为简单类型常数的常数表达式。

可使用文字初始化简单类型。例如，“A”是 [char](#) 类型的文字，而 2001 是 [int](#) 类型的文字。

初始化值类型

C# 中的局部变量经初始化后方可使用。因此，如果像下面这样声明了一个局部变量而未将其初始化：

```
int myInt;
```

那么在将其初始化之前，无法使用此变量。可使用下列语句将其初始化：

```
myInt = new int(); // Invoke default constructor for int type.
```

此语句等效于：

```
myInt = 0;           // Assign an initial value, 0 in this example.
```

当然，可以像下面这样用同一个语句进行声明和初始化：

```
int myInt = new int();
```

- 或 -

```
int myInt = 0;
```

使用 `new` 运算符时，将调用特定类型的默认构造函数并对变量赋以默认值。在上例中，默认构造函数将值 0 赋给了 `myInt`。有关通过调用默认构造函数所赋的值的更多信息，请参见[默认值表](#)。

对于用户定义的类型，使用 `new` 来调用默认构造函数。例如，下列语句调用了 `Point` 结构的默认构造函数：

```
Point p = new Point(); // Invoke default constructor for the struct.
```

此调用后，该结构被认为已被明确赋值；也就是说该结构的所有成员均已初始化为各自的默认值。

有关 `new` 运算符的更多信息，请参见[new](#)。

有关格式化数字类型输出的信息，请参见[格式化数值结果表](#)。

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[类型\(C# 参考\)](#)

[类型参考表\(C# 参考\)](#)

bool(C# 参考)

bool 关键字是 [System.Boolean](#) 的别名。它用于声明变量来存储布尔值 **true** 和 **false**。

注意

如果需要一个也可以有 **null** 值的布尔型变量，请使用 **bool**。有关更多信息，请参见[可空类型\(C# 编程指南\)](#)。

标识符

可将布尔值赋给 **bool** 变量。也可以将计算为 **bool** 类型的表达式赋给 **bool** 变量。

```
// keyword_bool.cs
using System;
public class MyClass
{
    static void Main()
    {
        bool i = true;
        char c = '0';
        Console.WriteLine(i);
        i = false;
        Console.WriteLine(i);

        bool Alphabetic = (c > 64 && c < 123);
        Console.WriteLine(Alphabetic);
    }
}
```

输出

```
True
False
False
```

转换

在 C++ 中，**bool** 类型的值可转换为 **int** 类型的值；也就是说，**false** 等效于零值，而 **true** 等效于非零值。在 C# 中，不存在 **bool** 类型与其他类型之间的相互转换。例如，下列 **if** 语句在 C# 中是非法的，而在 C++ 中则是合法的：

```
int x = 123;
if (x) // Invalid in C#
{
    printf_s("The value of x is nonzero.");
}
```

若要测试 **int** 类型的变量，必须将该变量与一个值（例如零）进行显式比较，如下所示：

```
int x = 123;
if (x != 0) // The C# way
{
    Console.Write("The value of x is nonzero.");
}
```

示例

在此例中，您从键盘输入一个字符，然后程序检查输入的字符是否是一个字母。如果输入的字符是字母，则程序检查是大写还是小写。这些检查是使用 [IsLetter](#) 和 [IsLower](#)（两者均返回 **bool** 类型）来执行的：

```
// keyword_bool_2.cs
using System;
public class BoolTest
{
```

```
static void Main()
{
    Console.Write("Enter a character: ");
    char c = (char)Console.Read();
    if (Char.IsLetter(c))
    {
        if (Char.IsLower(c))
        {
            Console.WriteLine("The character is lowercase.");
        }
        else
        {
            Console.WriteLine("The character is uppercase.");
        }
    }
    else
    {
        Console.WriteLine("Not an alphabetic character.");
    }
}
```

输入

X

示例输出

```
Enter a character: X
The character is uppercase.
Additional sample runs might look as follow:
Enter a character: x
The character is lowercase.
```

```
Enter a character: 2
The character is not an alphabetic character.
```

C# 语言规范

有关 **bool** 和相关主题的更多信息, 请参见 [C# 语言规范](#) 中的以下各节:

- 4.1.8 **bool** 类型
- 7.9.4 布尔相等运算符
- 7.11.1 布尔条件逻辑运算符

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

byte(C# 参考)

byte 关键字代表一种整型，该类型按下列所示存储值：

类型	范围	大小	.NET Framework 类型
byte	0 到 255	无符号 8 位整数	System.Byte

标识符

可如下例所示声明并初始化 **byte** 类型的变量：

```
byte myByte = 255;
```

在以上声明中，整数 255 从 **int** 隐式转换为 **byte**。如果整数超出了 **byte** 的范围，将产生编译错误。

转换

存在从 **byte** 到 **short**、**ushort**、**int**、**uint**、**long**、**ulong**、**float**、**double** 或 **decimal** 的预定义隐式转换。

不能将更大存储大小的非文本数值类型隐式转换为 **byte**。有关整型的存储大小的更多信息，请参见[整型表\(C# 参考\)](#)。例如，请看以下两个 **byte** 变量 **x** 和 **y**：

```
byte x = 10, y = 20;
```

以下赋值语句将产生一个编译错误，原因是赋值运算符右侧的算术表达式在默认情况下的计算结果为 **int** 类型。

```
// Error: conversion from int to byte:  
byte z = x + y;
```

若要解决此问题，请使用强制转换：

```
// OK: explicit conversion:  
byte z = (byte)(x + y);
```

但是，在目标变量具有相同或更大的存储大小时，使用下列语句是可能的：

```
int x = 10, y = 20;  
int m = x + y;  
long n = x + y;
```

同样，不存在从浮点型到 **byte** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
// Error: no implicit conversion from double:  
byte x = 3.0;  
// OK: explicit conversion:  
byte y = (byte)3.0;
```

调用重载方法时，必须使用显式转换。下面使用 **byte** 和 **int** 参数的重载方法为例：

```
public static void SampleMethod(int i) {}  
public static void SampleMethod(byte b) {}
```

使用 **byte** 强制转换可保证调用正确的类型，例如：

```
// Calling the method with the int parameter:  
SampleMethod(5);  
// Calling the method with the byte parameter:
```

```
SampleMethod((byte)5);
```

有关兼用浮点型和整型的算术表达式的信息，请参见 [float](#) 和 [double](#)。

有关隐式数值转换规则的更多信息，请参见[隐式数值转换表\(C# 参考\)](#)。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Byte](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

char(C# 参考)

char 关键字用于声明下表所示范围内的 Unicode 字符。Unicode 字符是 16 位字符，用于表示世界上多数已知的书面语言。

类型	范围	大小	.NET Framework 类型
char	U+0000 到 U+ffff	16 位 Unicode 字符	System.Char

标识符

char 类型的常数可以写成字符、十六进制换码序列或 Unicode 表示形式。您也可以显式转换整数字符代码。以下所有语句均声明了一个 **char** 变量并用字符 x 将其初始化：

```
char char1 = 'Z';           // Character literal
char char2 = '\x0058';      // Hexadecimal
char char3 = (char)88;       // Cast from integral type
char char4 = '\u0058';       // Unicode
```

转换

char 可以隐式转换为 [ushort](#)、[int](#)、[uint](#)、[long](#)、[ulong](#)、[float](#)、[double](#) 或 [decimal](#)。但是，不存在从其他类型到 **char** 类型的隐式转换。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 2.4.4.4 字符
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Char Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

decimal(C# 参考)

decimal 关键字表示 128 位数据类型。同浮点型相比, **decimal** 类型具有更高的精度和更小的范围, 这使它适合于财务和货币计算。**decimal** 类型的大致范围和精度如下表所示。

类型	大致范围	精度	.NET Framework 类型
decimal	$\pm 1.0 \times 10^{-28}$ 到 $\pm 7.9 \times 10^{28}$	28 到 29 位有效位	System.Decimal

标识符

如果希望实数被视为 **decimal** 类型, 请使用后缀 m 或 M, 例如:

```
decimal myMoney = 300.5m;
```

如果没有后缀 m, 数字将被视为 **double** 类型, 从而导致编译器错误。

转换

整型被隐式转换为 **decimal**, 其计算结果为 **decimal**。因此, 可以用整数初始化十进制变量而不使用后缀, 如下所示:

```
decimal myMoney = 300;
```

在浮点型和 **decimal** 类型之间不存在隐式转换;因此, 必须使用强制转换在这两种类型之间进行转换。例如:

```
decimal myMoney = 99.9m;
double x = (double)myMoney;
myMoney = (decimal)x;
```

还可以在同一表达式中混合使用 **decimal** 和数值整型。但是, 不进行强制转换就混合使用 **decimal** 和浮点型将导致编译错误。

有关隐式数值转换的更多信息, 请参见[隐式数值转换表\(C# 参考\)](#)。

有关显式数值转换的更多信息, 请参见[显式数值转换表\(C# 参考\)](#)。

格式化十进制输出

可以通过使用 **String.Format** 方法或 **System.Console.WriteLine** 方法(它调用 **String.Format()**)来格式化结果。指定货币格式时需要使用标准货币格式字符串“C”或“c”, 如示例 2 所示。有关 **String.Format** 方法的更多信息, 请参见[System.String.Format\(System.String,System.Object\)](#)。

示例

在此例中, 同一个表达式中混合使用了 **decimal** 和 **int**。计算结果为 **decimal** 类型。

如果试图使用下面这样的语句添加 **double** 和 **decimal** 变量:

```
double x = 9;
Console.WriteLine(d + x); // Error
```

将导致下列错误:

```
Operator '+' cannot be applied to operands of type 'double' and 'decimal'
```

```
// keyword_decimal.cs
// decimal conversion
using System;
public class TestDecimal
{
    static void Main ()
    {
```

```
    decimal d = 9.1m;
    int y = 3;
    Console.WriteLine(d + y);    // Result converted to decimal
}
}
```

输出

12.1

在此例中，输出用货币格式字符串格式化。注意：其中 `x` 被舍入，因为其小数点位置超出了 \$0.99。而表示最大精确位数的变量 `y` 严格按照正确的格式显示。

输出

My amount = \$1.00
Your amount = \$9,999,999,999,999,999,999,999,999,999,999,999,999.00

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
 - 4.1.7 decimal 类型

请参见

参考

C# 关键字

整型表(C# 参考)

内置类型表(C# 参考)

隐式数值转换表(C# 参考)

显式数值转换表(C# 参考)

Decimal Structure

概念

C# 编

其他资源

C# 参考

double(C# 参考)

double 关键字表示存储 64 位浮点值的简单类型。下表显示了 **double** 类型的精度和大致范围。

类型	大致范围	精度	.NET Framework 类型
double	$\pm 5.0 \times 10^{-324}$ 到 $\pm 1.7 \times 10^{308}$	15 到 16 位	System.Double

标识符

默认情况下，赋值运算符右侧的实数被视为 **double**。但是，如果希望整数被视为 **double**，请使用后缀 d 或 D，例如：

```
double x = 3D;
```

转换

可在一个表达式中兼用数值整型和浮点型。在此情况下，整型将转换为浮点型。根据以下规则计算表达式：

- 如果其中一个浮点类型为 **double**，则表达式的计算结果为 **double** 或 **bool**（对于关系表达式或布尔表达式）。
- 如果表达式中不存在 **double** 类型，则表达式的计算结果为 **float** 类型，在关系表达式或布尔表达式中为 **bool** 类型。

浮点表达式可以包含下列值集：

- 正零和负零。
- 正无穷和负无穷。
- 非数字值 (NaN)。
- 有限的非零值集。

有关这些值的更多信息，请参考“二进制浮点运算的 IEEE 标准”，该标准可在网站 <http://www.ieee.org/portal/index.jsp> 上获得。

示例

在下面的示例中，一个 **int**、一个 **short**、一个 **float** 和一个 **double** 相加，计算结果为 **double** 类型。

```
// keyword_double.cs
// Mixing types in expressions
using System;
class MixedTypes
{
    static void Main()
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        double w = 1.7E+3;
        // Result of the 2nd argument is a double:
        Console.WriteLine("The sum is {0}", x + y + z + w);
    }
}
```

输出

The sum is 1712.5

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

[请参见](#)

[参考](#)

[C# 关键字](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[默认值表\(C# 参考\)](#)

[浮点型表\(C# 参考\)](#)

enum(C# 参考)

enum 关键字用于声明枚举，即一种由一组称为枚举数列表的命名常数组成的独特类型。每种枚举类型都有基础类型，该类型可以是除 **char** 以外的任何整型。枚举元素的默认基础类型为 **int**。默认情况下，第一个枚举数的值为 0，后面每个枚举数的值依次递增 1。例如：

```
enum Days {Sat, Sun, Mon, Tue, Wed, Thu, Fri};
```

在此枚举中，`Sat` 为 0，`Sun` 为 1，`Mon` 为 2，依此类推。枚举数可以具有重写默认值的初始值设定项。例如：

```
enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};
```

在此枚举中，强制元素序列从 1 而不是 0 开始。

可以给 `Days` 类型的变量赋以基础类型范围内的任何值，所赋的值不限于已命名的常数。

enum E 的默认值为表达式 (E) 0 产生的值。

注意

枚举数的名称中不能包含空白。

基础类型指定为每个枚举数分配的存储大小。但是，从 **enum** 类型到整型的转换需要用显式类型转换来完成。例如，下面的语句通过使用强制转换从 **enum** 转换为 **int**，将枚举数 `Sun` 赋给 **int** 类型的变量：

```
int x = (int)Days.Sun;
```

将 **System.FlagsAttribute** 应用于某个枚举时，如果该枚举包含一些使用按位“或”运算组合的元素，这时您会注意到该属性在用于某些工具时会影响 **enum** 的行为。当使用诸如 **Console** 类方法、表达式计算器这样的工具时，可以注意到这些变化。（请参见示例 3）。

可靠编程

如果给新版本的枚举赋其他值，或者更改新版本中枚举成员的值，可能引起相关源代码的问题。情况通常是：**switch** 语句中使用了 **enum** 值，如果已将其他元素添加到 **enum** 类型中，那么默认值的测试可能意外地返回 `true`。

如果其他开发人员将使用您的代码，您需要提供相关说明，告诉开发人员将新元素添加到任何 **enum** 类型时，他们的代码应该如何响应。

示例

在此例中，声明了一个枚举 `Days`。两个枚举数被显式转换为整数并赋给整型变量。

```
// keyword_enum.cs
// enum initialization:
using System;
public class EnumTest
{
    enum Days {Sat=1, Sun, Mon, Tue, Wed, Thu, Fri};

    static void Main()
    {
        int x = (int)Days.Sun;
        int y = (int)Days.Fri;
        Console.WriteLine("Sun = {0}", x);
        Console.WriteLine("Fri = {0}", y);
    }
}
```

输出

```
Sun = 2
Fri = 7
```

在此例中，使用了基类选项来声明成员类型是 **long** 的 **enum**。注意，即使枚举的基础类型是 **long**，枚举成员仍然必须使用强制转换显式转换为 **long** 类型。

```
// keyword_enum2.cs
// Using long enumerators
using System;
public class EnumTest
{
    enum Range : long {Max = 2147483648L, Min = 255L};
    static void Main()
    {
        long x = (long)Range.Max;
        long y = (long)Range.Min;
        Console.WriteLine("Max = {0}", x);
        Console.WriteLine("Min = {0}", y);
    }
}
```

输出

```
Max = 2147483648
Min = 255
```

下面的代码示例阐释 **enum** 声明上的 **System.FlagsAttribute** 属性的使用和效果。

```
// enumFlags.cs
// Using the FlagsAttribute on enumerations.
using System;

[Flags]
public enum CarOptions
{
    SunRoof = 0x01,
    Spoiler = 0x02,
    FogLights = 0x04,
    TintedWindows = 0x08,
}

class FlagTest
{
    static void Main()
    {
        CarOptions options = CarOptions.SunRoof | CarOptions.FogLights;
        Console.WriteLine(options);
        Console.WriteLine((int)options);
    }
}
```

输出

```
SunRoof, FogLights
5
```

注释

注意：如果从 `Sat=1` 中移除初始值设定项，结果将是：

```
Sun = 1
Fri = 6
```

注释

请注意，如果移除 **FlagsAttribute**，此示例的输出为：

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.10 枚举
- 6.2.2 显式枚举转换
- 14 枚举

请参见

任务

[“属性”示例](#)

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

概念

[C# 编程指南](#)

[枚举设计](#)

其他资源

[C# 参考](#)

float(C# 参考)

float 关键字表示存储 32 位浮点值的简单类型。下表显示了 **float** 类型的精度和大致范围。

类型	大致范围	精度	.NET Framework 类型
float	$\pm 1.5 \times 10^{-45}$ 到 $\pm 3.4 \times 10^{38}$	7 位	System.Single

标识符

默认情况下，赋值运算符右侧的实数被视为 [double](#)。因此，应使用后缀 `f` 或 `F` 初始化浮点型变量，如下所示：

```
float x = 3.5F;
```

如果在以上声明中不使用后缀，则会因为您试图将一个 [double](#) 值存储到 **float** 变量中而发生编译错误。

转换

可在一个表达式中兼用数值整型和浮点型。在此情况下，整型将转换为浮点型。根据以下规则计算表达式：

- 如果其中一个浮点型为 [double](#)，则表达式的计算结果为 [double](#) 类型，在关系表达式或布尔表达式中为 [bool](#) 类型。
- 如果表达式中不存在 [double](#) 类型，则表达式的计算结果为 **float** 类型，在关系表达式或布尔表达式中为 [bool](#) 类型。

浮点表达式可以包含下列值集：

- 正零和负零
- 正无穷和负无穷
- 非数字值 (NaN)
- 有限的非零值集

有关这些值的更多信息，请参考“二进制浮点算法的 IEEE 标准”，该标准可在网站 <http://www.ieee.org/> 上获得。

示例

在下面的示例中，包含 [int](#)、[short](#) 和 **float** 类型的数学表达式得到一个 **float** 结果。请注意，表达式中没有 [double](#)。

```
// keyword_float.cs
// Mixing types in expressions
using System;
class MixedTypes
{
    static void Main()
    {
        int x = 3;
        float y = 4.5f;
        short z = 5;
        Console.WriteLine("The result is {0}", x * y / z);
    }
}
```

输出

The result is 2.7

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 4.1.6 浮点型
- 6.2.1 显式数值转换

[请参见](#)

[参考](#)

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Single Structure](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

int(C# 参考)

int 关键字表示一种整型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
int	-2,147,483,648 到 2,147,483,647	有符号 32 位整数	System.Int32

标识符

可以声明并初始化 **int** 类型的变量，例如：

```
int i = 123;
```

如果整数没有后缀，则其类型为以下类型中可表示其值的第一个类型：[int](#)、[uint](#)、[long](#)、[ulong](#)。在此例中为 **int** 类型。

转换

存在从 **int** 到 [long](#)、[float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。例如：

```
// '123' is an int, so an implicit conversion takes place here:
float f = 123;
```

存在从 [sbyte](#)、[byte](#)、[short](#)、[ushort](#) 或 [char](#) 到 **int** 的预定义隐式转换。例如，如果不进行强制转换，下面的赋值语句将产生编译错误：

```
long aLong = 22;
int i1 = aLong;          // Error: no implicit conversion from long.
int i2 = (int)aLong;    // OK: explicit conversion.
```

还请注意，不存在从浮点型到 **int** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
int x = 3.0;           // Error: no implicit conversion from double.
int y = (int)3.0;      // OK: explicit conversion.
```

有关兼用浮点型和整型的算术表达式的更多信息，请参见 [float](#) 和 [double](#)。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Int32 Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

long(C# 参考)

long 关键字表示一种整型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
long	-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807	有符号 64 位整数	System.Int64

标识符

可如下例所示声明并初始化 **long** 类型的变量：

```
long long1 = 4294967296;
```

如果整数没有后缀，则其类型为以下类型中可表示其值的第一个类型：[int](#)、[uint](#)、**long**、[ulong](#)。在上例中，它是 **long** 类型，因为它超出了 [uint](#) 的范围（有关整型的存储大小，请参见[整型表\(C# 参考\)](#)）。

还可以像下面这样，在 **long** 类型中使用后缀 L：

```
long long2 = 4294967296L;
```

当使用后缀 L 时，将根据整数的大小确定它的类型为 **long** 还是 [ulong](#)。在此例中，它是 **long**，因为它小于 [ulong](#) 的范围的下限。

此后缀常用于调用重载方法。以下面使用 **long** 和 [int](#) 参数的重载方法为例：

```
public static void SampleMethod(int i) {}
public static void SampleMethod(long l) {}
```

使用后缀 L 可保证调用正确的类型，例如：

```
SampleMethod(5);      // Calling the method with the int parameter
SampleMethod(5L);     // Calling the method with the long parameter
```

可在同一个表达式中同时使用 **long** 类型和其他数值整型，这时表达式的计算结果为 **long**（在关系表达式或布尔表达式中为 [bool](#)）类型。例如，下列表达式计算为 **long**：

```
898L + 88
```

注意

也可用小写字母“l”作后缀。但是，因为字母“l”容易与数字“1”混淆，会生成编译器警告。为清楚起见，请使用“L”。

有关兼用浮点型和整型的算术表达式的信息，请参见 [float](#) 和 [double](#)。

转换

存在从 **long** 到 [float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。其他情况下必须使用显式转换。例如，不使用显式类型转换时，下列语句将产生编译错误：

```
int x = 8L;          // Error: no implicit conversion from long to int
int x = (int)8L;    // OK: explicit conversion to int
```

存在从 [sbyte](#)、[byte](#)、[short](#)、[ushort](#)、[int](#)、[uint](#) 或 [char](#) 到 **long** 的预定义隐式转换。

还请注意，不存在从浮点型到 **long** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
long x = 3.0;           // Error: no implicit conversion from double
long y = (long)3.0;    // OK: explicit conversion
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Int64 Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

sbyte(C# 参考)

sbyte 关键字表示一种整型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
sbyte	-128 到 127	有符号 8 位整数	System.SByte

标识符

可如下例所示声明并初始化 **sbyte** 类型的变量：

```
sbyte sByte1 = 127;
```

在以上声明中，整数 127 从 [int](#) 隐式转换为 **sbyte**。如果整数超出了 **sbyte** 的范围，将产生编译错误。

调用重载方法时必须使用强制转换。以下面使用 **sbyte** 和 [int](#) 参数的重载方法为例：

```
public static void SampleMethod(int i) {}
public static void SampleMethod(sbyte b) {}
```

使用 **sbyte** 强制转换可保证调用正确的类型，例如：

```
// Calling the method with the int parameter:
SampleMethod(5);
// Calling the method with the sbyte parameter:
SampleMethod((sbyte)5);
```

转换

存在从 **sbyte** 到 [short](#)、[int](#)、[long](#)、[float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。

不能将存储大小更大的非文本数值类型隐式转换为 **sbyte** 类型（有关整型的存储大小的信息，请参见[整型表\(C# 参考\)](#)）。例如，请看以下两个 **sbyte** 变量 *x* 和 *y*：

```
sbyte x = 10, y = 20;
```

以下赋值语句将产生一个编译错误，原因是赋值运算符右侧的算术表达式在默认情况下的计算结果为 [int](#) 类型。

```
sbyte z = x + y; // Error: conversion from int to sbyte
```

若要解决此问题，请按如下方法对表达式进行强制转换：

```
sbyte z = (sbyte)(x + y); // OK: explicit conversion
```

但是，在目标变量具有相同或更大的存储大小时，使用下列语句是可能的：

```
sbyte x = 10, y = 20;
int m = x + y;
long n = x + y;
```

还请注意，不存在从浮点型到 **sbyte** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
sbyte x = 3.0;           // Error: no implicit conversion from double
sbyte y = (sbyte)3.0;    // OK: explicit conversion
```

有关兼用浮点型和整型的算术表达式的信息, 请参见 [float 和 double](#)。

有关隐式数值转换规则的更多信息, 请参见[隐式数值转换表\(C# 参考\)](#)。

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[SByte Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

short(C# 参考)

short 关键字表示一种整数数据类型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
short	-32,768 到 32,767	有符号 16 位整数	System.Int16

标识符

可如下例所示声明并初始化 **short** 类型的变量：

```
short x = 32767;
```

在以上声明中，整数 32767 从 [int](#) 隐式转换为 **short**。如果整数的长度超过了 **short** 存储位置的大小，则将产生编译错误。

调用重载方法时必须使用强制转换。以下面使用 **short** 和 [int](#) 参数的重载方法为例：

```
public static void SampleMethod(int i) {}
public static void SampleMethod(short s) {}
```

使用 **short** 强制转换可保证调用正确的类型，例如：

```
SampleMethod(5);           // Calling the method with the int parameter
SampleMethod((short)5);    // Calling the method with the short parameter
```

转换

存在从 **short** 到 [int](#)、[long](#)、[float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。

不能将存储大小更大的非文本数值类型隐式转换为 **short**（有关整型的存储大小的信息，请参见[整型表\(C# 参考\)](#)）。例如，请看以下两个 **short** 变量 **x** 和 **y**：

```
short x = 5, y = 12;
```

以下赋值语句将产生一个编译错误，原因是赋值运算符右侧的算术表达式在默认情况下的计算结果为 [int](#) 类型。

```
short z = x + y; // Error: no conversion from int to short
```

若要解决此问题，请使用强制转换：

```
short z = (short)(x + y); // OK: explicit conversion
```

但是，在目标变量具有相同或更大的存储大小时，使用下列语句是可能的：

```
int m = x + y;
long n = x + y;
```

不存在从浮点型到 **short** 的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
short x = 3.0;           // Error: no implicit conversion from double
short y = (short)3.0;    // OK: explicit conversion
```

有关兼用浮点型和整型的算术表达式的信息，请参见[float](#) 和 [double](#)。

有关隐式数值转换规则的更多信息，请参见[隐式数值转换表\(C# 参考\)](#)。

[C# 语言规范](#)

有关更多信息，请参见[C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量

- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[Int16](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

struct(C# 参考)

struct 类型是一种值类型，通常用来封装小型相关变量组，例如，矩形的坐标或库存商品的特征。下面的示例显示了一个简单的结构声明。

```
public struct Book
{
    public decimal price;
    public string title;
    public string author;
}
```

备注

结构还可以包含[构造函数](#)、[常量](#)、[字段](#)、[方法](#)、[属性](#)、[索引器](#)、[运算符](#)、[事件](#)和[嵌套类型](#)，但如果同时需要上述几种成员，则应当考虑改为使用类作为类型。

结构可以实现接口，但它们无法继承另一个结构。因此，结构成员无法声明为 **protected**。

有关更多信息，请参见[结构\(C# 编程指南\)](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 11 结构

请参见

参考

[C# 关键字](#)

[内置类型表\(C# 参考\)](#)

[值类型\(C# 参考\)](#)

[class\(C# 参考\)](#)

[接口\(C# 参考\)](#)

[对象、类和结构\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[默认值表\(C# 参考\)](#)

[类型\(C# 参考\)](#)

uint(C# 参考)

uint 关键字表示一种整型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
uint	0 到 4,294,967,295	无符号 32 位整数	System.UInt32

标识符

可如下例所示声明并初始化 **uint** 类型的变量：

```
uint myUint = 4294967290;
```

如果整数没有后缀，则其类型为以下类型中可表示其值的第一个类型：[int](#)、[uint](#)、[long](#)、[ulong](#)。在此例中，它是 **uint**：

```
uint uInt1 = 123;
```

还可以像下面这样使用后缀 **u** 或 **U**：

```
uint uInt2 = 123U;
```

当使用后缀 **U** 或 **u** 时，将根据文本的数值来确定文本的类型是 **uint** 还是 **ulong**。例如：

```
Console.WriteLine(44U.GetType());
Console.WriteLine(323442434344U.GetType());
```

此代码先后显示 [System.UInt32](#) 和 [System.UInt64](#)(它们分别是 **uint** 和 **ulong** 的基础类型)，因为第二个文本太大，无法用 **uint** 类型来存储。

转换

存在从 **uint** 到 [long](#)、[ulong](#)、[float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。例如：

```
float myFloat = 4294967290; // OK: implicit conversion to float
```

存在从 [byte](#)、[ushort](#) 或 [char](#) 到 **uint** 的预定义隐式转换。否则必须使用显式转换。例如，如果不进行强制转换，下面的赋值语句将产生编译错误：

```
long aLong = 22;
// Error -- no implicit conversion from long:
uint uInt1 = aLong;
// OK -- explicit conversion:
uint uInt2 = (uint)aLong;
```

还请注意，不存在从浮点型到 **uint** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
// Error -- no implicit conversion from double:
uint x = 3.0;
// OK -- explicit conversion:
uint y = (uint)3.0;
```

有关兼用浮点型和整型的算术表达式的信息，请参见 [float](#) 和 [double](#)。

有关隐式数值转换规则的更多信息，请参见[隐式数值转换表\(C# 参考\)](#)。

C# 语言规范

有关更多信息，请参见 C# 语言规范 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

C# 关键字

整型表(C# 参考)

内置类型表(C# 参考)

隐式数值转换表(C# 参考)

显式数值转换表(C# 参考)

UInt32 Structure

概念

C# 编程指南

其他资源

C# 参考

ulong(C# 参考)

ulong 关键字表示一种整型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
ulong	0 到 18,446,744,073,709,551,615	无符号 64 位整数	System.UInt64

标识符

可如下例所示声明并初始化 **ulong** 类型的变量：

```
ulong uLong = 9223372036854775808;
```

如果整数没有后缀，则其类型为以下类型中可表示其值的第一个类型：[int](#)、[uint](#)、[long](#)、[ulong](#)。在上面的示例中，它是 **ulong** 类型。

还可根据以下规则使用后缀指定文字类型：

- 如果使用 L 或 l，那么根据整数的大小，可以判断出其类型为 [long](#) 还是 **ulong**。

注意

注意也可用小写字母“l”作后缀。但是，因为字母“l”容易与数字“1”混淆，会生成编译器警告。为清楚起见，请使用“L”。

- 如果使用 U 或 u，那么根据整数的大小，可以判断出其类型为 [uint](#) 还是 **ulong**。
- 如果使用 UL、ul、Ul、uL、LU、lu、Lu 或 IU，则整数的类型为 **ulong**。

例如，以下三个语句的输出将为系统类型 [UInt64](#)，此类型对应于别名 **ulong**：

```
Console.WriteLine(9223372036854775808L.GetType());
Console.WriteLine(123UL.GetType());
Console.WriteLine((123UL + 456).GetType());
```

此后缀常用于调用重载方法。以下面使用 **ulong** 和 [int](#) 参数的重载方法为例：

```
public static void SampleMethod(int i) {}
public static void SampleMethod(ulong l) {}
```

在 **ulong** 参数后加上后缀可保证调用正确的类型，例如：

```
SampleMethod(5);      // Calling the method with the int parameter
SampleMethod(5UL);    // Calling the method with the ulong parameter
```

转换

存在从 **ulong** 到 [float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。

不存在从 **ulong** 到任何整型的隐式转换。例如，不使用显式类型转换时，下列语句将产生编译错误：

```
long long1 = 8UL;    // Error: no implicit conversion from ulong
```

存在从 [byte](#)、[ushort](#)、[uint](#) 或 [char](#) 到 **ulong** 的预定义隐式转换。

同样，不存在从浮点型到 **ulong** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
// Error -- no implicit conversion from double:
ulong x = 3.0;
```

```
// OK -- explicit conversion:  
ulong y = (ulong)3.0;
```

有关兼用浮点型和整型的算术表达式的信息，请参见 [float](#) 和 [double](#)。

有关隐式数值转换规则的更多信息，请参见[隐式数值转换表\(C# 参考\)](#)。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量
- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[UInt64 Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

ushort(C# 参考)

ushort 关键字表示一种整数数据类型，该类型根据下表显示的大小和范围存储值。

类型	范围	大小	.NET Framework 类型
ushort	0 到 65,535	无符号 16 位整数	System.UInt16

标识符

可如下例所示声明并初始化 **ushort** 类型的变量：

```
ushort myShort = 65535;
```

在以上声明中，整数 65535 从 [int](#) 隐式转换为 **ushort**。如果整数超出了 **ushort** 的范围，将产生编译错误。

调用重载方法时必须使用强制转换。以下面使用 **ushort** 和 [int](#) 参数的重载方法为例：

```
public static void SampleMethod(int i) {}
public static void SampleMethod(ushort s) {}
```

使用 **ushort** 强制转换可保证调用正确的类型，例如：

```
// Calls the method with the int parameter:
SampleMethod(5);
// Calls the method with the ushort parameter:
SampleMethod((ushort)5);
```

转换

存在从 **ushort** 到 [int](#)、[uint](#)、[long](#)、[ulong](#)、[float](#)、[double](#) 或 [decimal](#) 的预定义隐式转换。

存在从 [byte](#) 或 [char](#) 到 **ushort** 的预定义隐式转换。其他情况下必须使用显式转换。例如，请看以下两个 **ushort** 变量 *x* 和 *y*：

```
ushort x = 5, y = 12;
```

以下赋值语句将产生一个编译错误，原因是赋值运算符右侧的算术表达式在默认情况下的计算结果为 [int](#) 类型。

```
ushort z = x + y; // Error: conversion from int to ushort
```

若要解决此问题，请使用强制转换：

```
ushort z = (ushort)(x + y); // OK: explicit conversion
```

但是，在目标变量具有相同或更大的存储大小时，使用下列语句是可能的：

```
int m = x + y;
long n = x + y;
```

还请注意，不存在从浮点型到 **ushort** 类型的隐式转换。例如，除非使用显式强制转换，否则以下语句将生成一个编译器错误：

```
// Error -- no implicit conversion from double:
ushort x = 3.0;
// OK -- explicit conversion:
ushort y = (ushort)3.0;
```

有关兼用浮点型和整型的算术表达式的信息，请参见 [float](#) 和 [double](#)。

有关隐式数值转换规则的更多信息，请参见 [隐式数值转换表\(C# 参考\)](#)。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.3 类型和变量

- 4.1.5 整型

请参见

参考

[C# 关键字](#)

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

[UInt16 Structure](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

引用类型(C# 参考)

引用类型的变量又称为对象，可存储对实际数据的引用。本节介绍以下用于声明引用类型的关键字：

- [class](#)
- [interface](#)
- [delegate](#)

本节也介绍以下内置引用类型：

- [object](#)
- [string](#)

请参见

参考

[C# 关键字](#)

[值类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[类型\(C# 参考\)](#)

class(C# 参考)

类是使用关键字 **class** 声明的，如下面的示例所示：

```
class TestClass
{
    // Methods, properties, fields, events, delegates
    // and nested classes go here.
}
```

备注

与 C++ 不同，C# 中仅允许单个继承。也就是说，类只能从一个基类继承实现。但是，一个类可以实现一个以上的接口。下表给出了类继承和接口实现的一些示例：

继承	示例
无	class ClassA { }
单个	class DerivedClass: BaseClass { }
无，实现两个接口	class ImplClass: IFace1, IFace2 { }
单一，实现一个接口	class ImplDerivedClass: BaseClass, IFace1 { }

只有嵌套类允许访问级别 [protected](#) 和 [private](#)。

还可以声明有类型参数的泛型类；有关更多信息，请参见[泛型类](#)。

一个类可包含下列成员的声明：

- [构造函数](#)
- [析构函数](#)
- [常数](#)
- [字段](#)
- [方法](#)
- [属性](#)
- [索引器](#)
- [运算符](#)
- [事件](#)
- [委托](#)
- [类](#)
- [接口](#)
- [结构](#)

示例

下面的示例说明如何声明类的字段、构造函数和方法。该例还说明了如何实例化对象及如何打印实例数据。在此例中声明了两个类，一个是 `Kid` 类，它包含两个私有字段(`name` 和 `age`)和两个公共方法。第二个类 `MainClass` 用来包含 `Main`。

```
// keyword_class.cs
// class example
using System;
```

```

class Kid
{
    private int age;
    private string name;

    // Default constructor:
    public Kid()
    {
        name = "N/A";
    }

    // Constructor:
    public Kid(string name, int age)
    {
        this.name = name;
        this.age = age;
    }

    // Printing method:
    public void PrintKid()
    {
        Console.WriteLine("{0}, {1} years old.", name, age);
    }
}

class MainClass
{
    static void Main()
    {
        // Create objects
        // Objects must be created using the new operator:
        Kid kid1 = new Kid("Craig", 11);
        Kid kid2 = new Kid("Sally", 10);

        // Create an object using the default constructor:
        Kid kid3 = new Kid();

        // Display results:
        Console.Write("Kid #1: ");
        kid1.PrintKid();
        Console.Write("Kid #2: ");
        kid2.PrintKid();
        Console.Write("Kid #3: ");
        kid3.PrintKid();
    }
}

```

输出

```

Kid #1: Craig, 11 years old.
Kid #2: Sally, 10 years old.
Kid #3: N/A, 0 years old.

```

注释

注意：在上例中，私有字段（`name` 和 `age`）只能通过 `Kid` 类的公共方法访问。例如，不能通过 `Main` 方法用如下语句打印小孩的姓名：

```
Console.WriteLine(kid1.name); // Error
```

只有当 `Main` 是 `Kid` 的成员时，才能从 `Main` 访问该类的私有成员。

如果在类的内部声明的类型没有访问修饰符，则该类型默认为 **private**，因此，如果移除关键字，则此示例中的数据成员仍然会是 **private** 的。

最后要注意的是，默认情况下，对于使用默认构造函数（`kid3`）创建的对象，`age` 字段初始化为零。

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6 类和对象
- 3.4.4 类成员
- 4.2.1 类类型
- 10 类

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

委托 (C# 参考)

委托类型声明的格式如下：

```
public delegate void TestDelegate(string message);
```

delegate 关键字用于声明一个引用类型，该引用类型可用于封装命名方法或匿名方法。委托类似于 C++ 中的函数指针；但是，委托是类型安全和可靠的。有关委托的应用，请参见[委托](#)和[泛型委托](#)。

备注

委托是[事件](#)的基础。

通过将委托与命名方法或匿名方法关联，可以实例化委托。有关更多信息，请参见[命名方法](#)和[匿名方法](#)。

为了与命名方法一起使用，委托必须用具有可接受签名的方法进行实例化。有关方法签名中允许的方差度的更多信息，请参见[委托中的协变和逆变](#)。为了与匿名方法一起使用，委托和与之关联的代码必须一起声明。本节讨论这两种实例化委托的方法。

```
using System;
// Declare delegate -- defines required signature:
delegate void SampleDelegate(string message);

class MainClass
{
    // Regular method that matches signature:
    static void SampleDelegateMethod(string message)
    {
        Console.WriteLine(message);
    }

    static void Main()
    {
        // Instantiate delegate with named method:
        SampleDelegate d1 = SampleDelegateMethod;
        // Instantiate delegate with anonymous method:
        SampleDelegate d2 = delegate(string message)
        {
            Console.WriteLine(message);
        };

        // Invoke delegate d1:
        d1("Hello");
        // Invoke delegate d2:
        d2(" World");
    }
}
```

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.11 委托
- 15 委托

请参见

参考

[C# 关键字](#)

[引用类型 \(C# 参考\)](#)

[匿名方法 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[委托 \(C# 编程指南\)](#)

[事件 \(C# 编程指南\)](#)

[命名方法 \(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

接口 (C# 参考)

接口只包含[方法、委托或事件](#)的签名。方法的实现是在实现接口的类中完成的，如下面的示例所示：

```

        interface ISampleInterface
    {
        void SampleMethod();
    }

    class ImplementationClass : ISampleInterface
    {
        // Explicit interface member implementation:
        void ISampleInterface.SampleMethod()
        {
            // Method implementation.
        }

        static void Main()
        {
            // Declare an interface instance.
            ISampleInterface obj = new ImplementationClass();

            // Call the member.
            obj.SampleMethod();
        }
    }
}

```

备注

接口可以是命名空间或类的成员，并且可以包含下列成员的签名：

- [方法](#)
- [属性](#)
- [索引器](#)
- [事件](#)

一个接口可从一个或多个基接口继承。

当基类型列表包含基类和接口时，基类必须是列表中的第一项。

实现接口的类可以显式实现该接口的成员。显式实现的成员不能通过类实例访问，而只能通过接口实例访问，例如：

[有关显式接口实现的更多详细信息和代码示例，请参见显式接口实现 \(C# 编程指南\)。](#)

示例

下面的示例演示了接口实现。在此例中，接口 `IPoint` 包含属性声明，后者负责设置和获取字段的值。`Point` 类包含属性实现。

```

// keyword_interface_2.cs
// Interface implementation
using System;
interface IPoint
{
    // Property signatures:
    int x
    {
        get;
        set;
    }

    int y
    {
        get;
    }
}

```

```

        set;
    }

}

class Point : IPoint
{
    // Fields:
    private int _x;
    private int _y;

    // Constructor:
    public Point(int x, int y)
    {
        _x = x;
        _y = y;
    }

    // Property implementation:
    public int x
    {
        get
        {
            return _x;
        }

        set
        {
            _x = value;
        }
    }

    public int y
    {
        get
        {
            return _y;
        }
        set
        {
            _y = value;
        }
    }
}

class MainClass
{
    static void PrintPoint(IPoint p)
    {
        Console.WriteLine("x={0}, y={1}", p.x, p.y);
    }

    static void Main()
    {
        Point p = new Point(2, 3);
        Console.Write("My Point: ");
        PrintPoint(p);
    }
}

```

输出

My Point: x=2, y=3

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.9 接口

- 3.4.5 接口成员
- 4.2.4 接口类型
- 10.1.2.2 接口实现
- 11.2 结构接口
- 13 接口

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

[使用属性\(C# 编程指南\)](#)

[使用索引器\(C# 编程指南\)](#)

[class\(C# 参考\)](#)

[struct\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

object(C# 参考)

object 类型在 .NET Framework 中是 [Object](#) 的别名。在 C# 的统一类型系统中，所有类型(预定义类型、用户定义类型、引用类型和值类型)都是直接或间接从 **Object** 继承的。可以将任何类型的值赋给 **object** 类型的变量。将值类型的变量转换为对象的过程称为“装箱”。将对象类型的变量转换为值类型的过程称为“取消装箱”。有关更多信息，请参见[装箱和取消装箱](#)。

示例

下面的示例演示了 **object** 类型的变量如何接受任何数据类型的值，以及 **object** 类型的变量如何在 .NET Framework 中使用 **Object** 的方法。

```
// keyword_object.cs
using System;
class SampleClass
{
    public int i = 10;
}

class MainClass
{
    static void Main()
    {
        object a;
        a = 1;      // an example of boxing
        Console.WriteLine(a);
        Console.WriteLine(a.GetType());
        Console.WriteLine(a.ToString());

        a = new SampleClass();
        SampleClass classRef;
        classRef = (SampleClass)a;
        Console.WriteLine(classRef.i);
    }
}
```

输出

```
1
System.Int32
1
10
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1 简介](#)
- [4.2.2 对象类型](#)

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

[值类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

string(C# 参考)

string 类型表示零或更多 Unicode 字符组成的序列。**string** 是 .NET Framework 中 [String](#) 的别名。

尽管 **string** 是引用类型，但定义相等运算符(== 和 !=)是为了比较 **string** 对象(而不是引用)的值。这使得对字符串相等性的测试更为直观。例如：

```
string a = "hello";
string b = "h";
// Append to contents of 'b'
b += "ello";
Console.WriteLine(a == b);
Console.WriteLine((object)a == (object)b);
```

这将先显示“True”，然后显示“False”，因为字符串的内容是相同的，但是 **a** 和 **b** 引用的不是同一个字符串实例。

+ 运算符用于连接字符串：

```
string a = "good " + "morning";
```

这将创建一个包含“good morning”的字符串对象。

字符串是不可变的，即：字符串对象在创建后，尽管从语法上看您似乎可以更改其内容，但事实上并不可行。例如，编写此代码时，编译器实际上会创建一个新字符串对象来保存新的字符序列，变量 **b** 继续保存“h”。

```
string b = "h";
b += "ello";
```

[] 运算符可以用来访问 **string** 中的各个字符：

```
string str = "test";
char x = str[2]; // x = 's';
```

字符串为 **string** 类型并可写成两种形式，即用引号引起起来和用 @ 引起来。用引号引起起来的字符串括在双引号 ("") 内：

```
"good morning" // a string literal
```

字符串可以包含包括转义序列在内的任何字符：

```
string a = "\\\u0066\n";
```

上面的字符串包含一个反斜杠、字母 f 和一个新行。

注意

转义码 \udddd(其中 dddd 是一个四位数)表示 Unicode 字符 U+dddd。此外还识别 8 位 Unicode 转义码:\udddd\udddd。

用 @ 引起来的字符串以 @ 开头，并且也用双引号引起起来。例如：

```
@"good morning" // a string literal
```

用 @ 引起来的优点在于换码序列“不”被处理，这样就可以轻松写出字符串，例如一个完全限定的文件名：

```
@"c:\Docs\Source\a.txt" // rather than "c:\\Docs\\Source\\a.txt"
```

若要在用 @ 引起来的字符串中包括一个双引号，请使用两对双引号：

```
""""Ahoy!"" cried the captain." // "Ahoy!" cried the captain.
```

@ 符号的另一种用法是使用碰巧成为 C# 关键字的被引用的 ([/reference](#)) 标识符。

示例

```
// keyword_string.cs
using System;
class TestClass
{
    static void Main()
    {
        string a = "\u0068ello ";
        string b = "world";
        Console.WriteLine( a + b );
        Console.WriteLine( a + b == "hello world" );
    }
}
```

输出

```
hello world
True
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 2.4.2 标识符
- 2.4.4.5 字符串
- 4.2.3 字符串类型
- 7.9.7 字符串相等运算符

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

[值类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

[C# 编程指南](#)

其他资源

[C# 参考](#)

[设置数值结果表的格式\(C# 参考\)](#)

void(C# 参考)

用作方法的返回类型时, **void** 关键字指定方法不返回值。

在方法的参数列表中不允许使用 **void**。采用以下形式声明一个无参数的、不返回值的方法:

```
void SampleMethod();
```

也可在不安全上下文中使用 **void** 将指针声明为未知类型。有关更多信息, 请参见[指针类型\(C# 编程指南\)](#)。

void 是 .NET Framework [System.Void](#) 类型的别名。

C# 语言规范

有关更多信息, 请参见[C# 语言规范](#)中的以下各章节:

- 1.6.5 方法

请参见

参考

[C# 关键字](#)

[引用类型\(C# 参考\)](#)

[值类型\(C# 参考\)](#)

[方法\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

类型参考表 (C# 参考)

以下参考表概括了 C# 的类型：

[内置类型表](#)

[整型](#)

[浮点型](#)

[默认值](#)

[值类型](#)

[隐式数值转换](#)

[显式数值转换表](#)

有关格式化数字类型输出的信息，请参见[格式化数值结果表](#)。

请参见

[参考](#)

[引用类型 \(C# 参考\)](#)

[值类型 \(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

内置类型表(C# 参考)

下表显示了内置 C# 类型的关键字，这些类型是 [System](#) 命名空间中的预定义类型的别名。

C# 类型	.NET Framework 类型
<code>bool</code>	System.Boolean
<code>byte</code>	System.Byte
<code>sbyte</code>	System.SByte
<code>char</code>	System.Char
<code>decimal</code>	System.Decimal
<code>double</code>	System.Double
<code>float</code>	System.Single
<code>int</code>	System.Int32
<code>uint</code>	System.UInt32
<code>long</code>	System.Int64
<code>ulong</code>	System.UInt64
<code>object</code>	System.Object
<code>short</code>	System.Int16
<code>ushort</code>	System.UInt16
<code>string</code>	System.String

备注

除了 `object` 和 `string` 外，表中的所有类型均称为简单类型。

C# 类型的关键字及其别名可以互换。例如，可使用下列两种声明中的一种来声明一个整数变量：

```
int x = 123;
System.Int32 x = 123;
```

若要显示任何 C# 类型的实际类型，请使用系统方法 `GetType()`。例如，下列语句显示了表示 `myVariable` 类型的系统别名：

```
Console.WriteLine(myVariable.GetType());
```

还可使用 `typeof` 运算符。

请参见

参考

[C# 关键字](#)

[值类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[默认值表\(C# 参考\)](#)

[设置数值结果表的格式\(C# 参考\)](#)

[类型参考表\(C# 参考\)](#)

整型表 (C# 参考)

下表显示了整型的大小和范围，这些类型构成了简单类型的一个子集。

类型	范围	大小
<code>sbyte</code>	-128 到 127	有符号 8 位整数
<code>byte</code>	0 到 255	无符号 8 位整数
<code>char</code>	U+0000 到 U+ffff	16 位 Unicode 字符
<code>short</code>	-32,768 到 32,767	有符号 16 位整数
<code>ushort</code>	0 到 65,535	无符号 16 位整数
<code>int</code>	-2,147,483,648 到 2,147,483,647	有符号 32 位整数
<code>uint</code>	0 到 4,294,967,295	无符号 32 位整数
<code>long</code>	-9,223,372,036,854,775,808 到 9,223,372,036,854,775,807	有符号 64 位整数
<code>ulong</code>	0 到 18,446,744,073,709,551,615	无符号 64 位整数

如果整数表示的值超出了 `ulong` 的范围，将产生编译错误。

[请参见](#)

[参考](#)

[C# 关键字](#)

[内置类型表 \(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[浮点型表 \(C# 参考\)](#)

[默认值表 \(C# 参考\)](#)

[设置数值结果表的格式 \(C# 参考\)](#)

[类型参考表 \(C# 参考\)](#)

浮点型表 (C# 参考)

下表显示了浮点型的精度和大致范围。

类型	大致范围	精度
float	$\pm 1.5e-45$ 到 $\pm 3.4e38$	7 位
double	$\pm 5.0e-324$ 到 $\pm 1.7e308$	15 到 16 位

请参见

参考

[内置类型表 \(C# 参考\)](#)

[整型表 \(C# 参考\)](#)

[decimal \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[默认值表 \(C# 参考\)](#)

[设置数值结果表的格式 \(C# 参考\)](#)

[类型参考表 \(C# 参考\)](#)

默认值表 (C# 参考)

下表显示了由默认构造函数返回的值类型的默认值。默认构造函数是通过 **new** 运算符来调用的，如下所示：

```
int myInt = new int();
```

以上语句同下列语句效果相同：

```
int myInt = 0;
```

请记住：在 C# 中不允许使用未初始化的变量。

值类型	默认值
bool	false
byte	0
char	'\0'
decimal	0.0M
double	0.0D
enum	表达式 (E)0 产生的值，其中 E 为 enum 标识符。
float	0.0F
int	0
long	0L
sbyte	0
short	0
struct	将所有的值类型字段设置为默认值并将所有的引用类型字段设置为 null 时产生的值。
uint	0
ulong	0
ushort	0

[请参见](#)

[参考](#)

[值类型 \(C# 参考\)](#)

[内置类型表 \(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[类型参考表 \(C# 参考\)](#)

值类型表(C# 参考)

下表按类别列出了 C# 的值类型。

值类型	类别
bool	布尔型
byte	无符号、数值、整数
char	无符号、数值、整数
decimal	数值、十进制
double	数值、浮点
enum	枚举
float	数值、浮点
int	有符号、数值、整数
long	有符号、数值、整数
sbyte	有符号、数值、整数
short	有符号、数值、整数
struct	用户定义的结构
uint	无符号、数值、整数
ulong	无符号、数值、整数
ushort	无符号、数值、整数

请参见

参考

[值类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[设置数值结果表的格式\(C# 参考\)](#)

[类型参考表\(C# 参考\)](#)

隐式数值转换表(C# 参考)

下表显示了预定义的隐式数值转换。隐式转换可能在多种情形下发生，包括调用方法时和在赋值语句中。

从	到
sbyte	short, int, long, float, double 或 decimal
byte	short, ushort, int, uint, long, ulong, float, double 或 decimal
short	int, long, float, double 或 decimal
ushort	int, uint, long, ulong, float, double 或 decimal
int	long, float, double 或 decimal
uint	long, ulong, float, double 或 decimal
long	float, double 或 decimal
char	ushort, int, uint, long, ulong, float, double 或 decimal
float	double
ulong	float, double 或 decimal

备注

- 从 **int, uint 或 long** 到 **float** 的转换以及从 **long** 到 **double** 的转换的精度可能会降低，但数值大小不受影响。
- 不存在到 **char** 类型的隐式转换。
- 不存在浮点型与 **decimal** 类型之间的隐式转换。
- int** 类型的常数表达式可转换为 **sbyte, byte, short, ushort, uint** 或 **ulong**，前提是常数表达式的值处于目标类型的范围之内。

C# 语言规范

有关更多信息，请参见 C# 语言规范([C# 语言规范](#))：

- 6.1 隐式转换
- 7.15 常数表达式

请参见

参考

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[显式数值转换表\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

显式数值转换表(C# 参考)

显式数值转换用于通过显式转换表达式，将任何数字类型转换为任何其他数字类型。对于它不存在隐式转换。下表显示了这些转换。

从	到
sbyte	byte 、 ushort 、 uint 、 ulong 或 char
byte	Sbyte 或者 char
short	sbyte 、 byte 、 ushort 、 uint 、 ulong 或 char
ushort	sbyte 、 byte 、 short 或 char
int	sbyte 、 byte 、 short 、 ushort 、 uint 、 ulong 或 char
uint	sbyte 、 byte 、 short 、 ushort 、 int 或 char
long	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 ulong 或 char
ulong	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 或 char
char	sbyte 、 byte 或 short
float	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 、 ulong 、 char 或 decimal
double	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 、 ulong 、 char 、 float 或 decimal
decimal	sbyte 、 byte 、 short 、 ushort 、 int 、 uint 、 long 、 ulong 、 char 、 float 或 double

备注

- 显式数值转换可能导致精度损失或引发异常。
- 将 **decimal** 值转换为整型时，该值将舍入为与零最接近的整数值。如果结果整数值超出目标类型的范围，则会引发 **OverflowException**。
- 将 **double** 或 **float** 值转换为整型时，值会被截断。如果该结果整数值超出了目标值的范围，其结果将取决于溢出检查上下文。在 checked 上下文中，将引发 **OverflowException**；而在 unchecked 上下文中，结果将是一个未指定的目标类型的值。
- 将 **double** 转换为 **float** 时，**double** 值将舍入为最接近的 **float** 值。如果 **double** 值因过小或过大而使目标类型无法容纳它，则结果将为零或无穷大。
- 将 **float** 或 **double** 转换为 **decimal** 时，源值将转换为 **decimal** 表示形式，并舍入为第 28 个小数位之后最接近的数（如果需要）。根据源值的不同，可能产生以下结果：
 - 如果源值因过小而无法表示为 **decimal**，那么结果将为零。
 - 如果源值为 NaN（非数字值）、无穷大或因过大而无法表示为 **decimal**，则会引发 **OverflowException**。
- 将 **decimal** 转换为 **float** 或 **double** 时，**decimal** 值将舍入为最接近的 **double** 或 **float** 值。

有关显式转换的更多信息，请参见“6.2 C# 语言规范中的显式”。有关如何访问此规范的更多信息，请参见 [C# 语言规范](#)。

请参见

参考

[整型表\(C# 参考\)](#)

[内置类型表\(C# 参考\)](#)

[隐式数值转换表\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

设置数值结果表的格式 (C# 参考)

可以通过使用 **String.Format** 方法或 **Console.WriteLine** 方法(它调用 **String.Format**)来设置数值结果的格式。通过格式字符串指定格式。下表包含受支持的标准格式字符串。格式字符串采用以下形式:*A**xx*, 其中 *A* 是格式说明符, *xx* 是精度说明符。格式说明符控制应用于数值的格式类型, 而精度说明符则控制格式化输出的有效位数或小数位数。

有关标准及自定义格式化字符串的更多信息, 请参见[格式化概述](#)。有关 **String.Format** 方法的更多信息, 请参见[System.String.Format](#)。

字符	说明	示例	输出
C 或 c	货币	Console.WriteLine("{0:C}", 2.5); Console.WriteLine("{0:C}", -2.5);	\$2.50 (\$2.50)
D 或 d	十进制数	Console.WriteLine("{0:D5}", 25);	00025
E 或 e	科学型	Console.WriteLine("{0:E}", 250000);	2.50000E+005
F 或 f	固定点	Console.WriteLine("{0:F2}", 25); Console.WriteLine("{0:F0}", 25);	25.00 25
G 或 g	常规	Console.WriteLine("{0:G}", 2.5);	2.5
N 或 n	数字	Console.WriteLine("{0:N}", 2500000);	2,500,000.00
X 或 x	十六进制	Console.WriteLine("{0:X}", 250); Console.WriteLine("{0:X}", 0xffff);	FA FFFF

[请参见](#)

[参考](#)

[string\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[类型参考表\(C# 参考\)](#)

修饰符(C# 参考)

修饰符用于修改类型和类型成员的声明。本节介绍 C# 修饰符：

修饰符	用途
访问修饰符	指定声明的类型和类型成员的可访问性。 <ul style="list-style-type: none"> • public • private • internal • protected
abstract	指示某个类只能是其他类的基类。
const	指定无法修改字段或局部变量的值。
event	声明事件。
extern	指示在外部实现方法。
• new	从基类成员隐藏继承的成员。
override	提供从基类继承的虚拟成员的新实现。
partial	在整个同一程序集中定义分部类和结构。
readonly	声明一个字段，该字段只能赋值为该声明的一部分或者在同一个类的构造函数中。
sealed	指定类不能被继承。
static	声明属于类型本身而不是属于特定对象的成员。
unsafe	声明不安全的上下文。
virtual	在派生类中声明其实现可由重写成员更改的方法或访问器。
volatile	指示字段可由操作系统、硬件或并发执行线程等在程序中进行修改。

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

访问修饰符(C# 参考)

访问修饰符是一些关键字，用于指定声明的成员或类型的可访问性。本节介绍四个访问修饰符：

- [public](#)
- [protected](#)
- [internal](#)
- [private](#)

使用这些访问修饰符可指定下列五个可访问性级别：

public: 访问不受限制。

protected: 访问仅限于包含类或从包含类派生的类型。

internal: 访问仅限于当前程序集。

protected internal: 访问仅限于当前程序集或从包含类派生的类型。

private: 访问仅限于包含类型。

本节还介绍以下内容：

- [可访问性级别](#): 使用四个访问修饰符声明五个可访问性级别。
- [可访问域](#): 指定在程序节的哪个位置可以引用成员。
- [可访问性级别的使用限制](#): 概述了声明的可访问性级别的使用限制。

请参见

参考

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

可访问性级别(C# 参考)

使用访问修饰符 `public`、`protected`、`internal` 或 `private` 可以为成员指定以下声明的可访问性之一。

声明的可访问性	含义
<code>public</code>	访问不受限制。
<code>protected</code>	访问仅限于包含类或从包含类派生的类型。
<code>internal</code>	访问仅限于当前程序集。
<code>protected internal</code>	访问仅限于从包含类派生的当前程序集或类型。
<code>private</code>	访问仅限于包含类型。

一个成员或类型只能有一个访问修饰符，使用 `protectedinternal` 组合时除外。

命名空间上不允许使用访问修饰符。命名空间没有访问限制。

根据发生成员声明的上下文，只允许某些声明的可访问性。如果在成员声明中未指定访问修饰符，则使用默认的可访问性。

不嵌套在其他类型中的顶级类型的可访问性只能是 `internal` 或 `public`。这些类型的默认可访问性是 `internal`。

嵌套类型是其他类型的成员，它们可以具有下表所示的声明的可访问性。

属于	默认的成员可访问性	该成员允许的声明的可访问性
<code>enum</code>	<code>public</code>	无
<code>class</code>	<code>private</code>	<code>public</code> <code>protected</code> <code>internal</code> <code>private</code> <code>protected internal</code>
<code>interface</code>	<code>public</code>	无
<code>struct</code>	<code>private</code>	<code>public</code> <code>internal</code> <code>private</code>

嵌套类型的可访问性取决于它的可访问域，该域是由已声明的成员可访问性和直接包含类型的可访问域这二者共同确定的。但是，嵌套类型的可访问域不能超出包含它的类型的可访问域。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.3 对实例成员的受保护访问
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性

请参见

参考

[C# 关键字](#)

[访问修饰符\(C# 参考\)](#)

[可访问域\(C# 参考\)](#)

[可访问性级别的使用限制\(C# 参考\)](#)

[访问修饰符\(C# 编程指南\)](#)

[public\(C# 参考\)](#)

[private\(C# 参考\)](#)

[protected\(C# 参考\)](#)

[internal\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

可访问域(C# 参考)

成员的可访问域指定程序段中可以引用成员的位置。如果成员嵌套在其他类型中，其可访问域由该成员的[可访问性级别](#)和直接包含类型的可访问域共同确定。

顶级类型的可访问域至少是声明它的项目的程序文本，即该项目的整个源文件。嵌套类型的可访问域至少是声明它的类型的程序文本，即包括任何嵌套类型的类型体。嵌套类型的可访问域决不能超出包含类型的可访问域。这些概念在以下示例中加以说明。

示例

该示例包含一个顶级类型 T1 和两个嵌套类 M1 和 M2。这两个类包含具有不同声明的可访问性的字段。在 Main 方法中，每个语句后都有注释，指示每个成员的可访问域。注意，试图引用不可访问的成员的语句被注释掉了。如果希望查看由引用不可访问的成员所导致的编译器错误，请逐个移除注释。

```
// cs_Accessibility_Domain.cs
using System;
namespace AccessibilityDomainNamespace
{
    public class T1
    {
        public static int publicInt;
        internal static int internalInt;
        private static int privateInt = 0; // CS0414

        public class M1
        {
            public static int publicInt;
            internal static int internalInt;
            private static int privateInt = 0; // CS0414
        }

        private class M2
        {
            public static int publicInt = 0;
            internal static int internalInt = 0;
            private static int privateInt = 0; // CS0414
        }
    }

    class MainClass
    {
        static void Main()
        {
            // Access is unlimited:
            T1.publicInt = 1;
            // Accessible only in current assembly:
            T1.internalInt = 2;
            // Error: inaccessible outside T1:
            //     T1.myPrivateInt = 3;

            // Access is unlimited:
            T1.M1.publicInt = 1;
            // Accessible only in current assembly:
            T1.M1.internalInt = 2;
            // Error: inaccessible outside M1:
            //     T1.M1.myPrivateInt = 3;

            // Error: inaccessible outside T1:
            //     T1.M2.myPublicInt = 1;
            // Error: inaccessible outside T1:
            //     T1.M2.myInternalInt = 2;
            // Error: inaccessible outside M2:
            //     T1.M2.myPrivateInt = 3;
        }
    }
}
```

}

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.2 可访问域
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性

请参见

参考

[C# 关键字](#)

[访问修饰符 \(C# 参考\)](#)

[可访问性级别 \(C# 参考\)](#)

[可访问性级别的使用限制 \(C# 参考\)](#)

[访问修饰符 \(C# 编程指南\)](#)

[public \(C# 参考\)](#)

[private \(C# 参考\)](#)

[protected \(C# 参考\)](#)

[internal \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

可访问性级别的使用限制 (C# 参考)

声明类型时，最重要的是查看该类型是否必须至少与其他成员或类型具有同样的可访问性。例如，直接基类必须至少与派生类具有同样的可访问性。以下声明将导致编译器错误，因为基类 `BaseClass` 的可访问性小于 `MyClass`：

```
class BaseClass {...}
public class MyClass: BaseClass {...} // Error
```

下表汇总了对使用声明的可访问性级别的限制。

上下文	备注
类	类类型的直接基类必须至少与类类型本身具有同样的可访问性。
接口	接口类型的显式基接口必须至少与接口类型本身具有同样的可访问性。
委托	委托类型的返回类型和参数类型必须至少与委托类型本身具有同样的可访问性。
常数	常数的类型必须至少与常数本身具有同样的可访问性。
字段	字段的类型必须至少与字段本身具有同样的可访问性。
方法	方法的返回类型和参数类型必须至少与方法本身具有同样的可访问性。
属性	属性的类型必须至少与属性本身具有同样的可访问性。
事件	事件的类型必须至少与事件本身具有同样的可访问性。
索引器	索引器的类型和参数类型必须至少与索引器本身具有同样的可访问性。
运算符	运算符的返回类型和参数类型必须至少与运算符本身具有同样的可访问性。
构造函数	构造函数的参数类型必须至少与构造函数本身具有同样的可访问性。

示例

下面的示例包含不同类型的错误声明。每个声明后的注释指示了预期的编译器错误。

```
// Restrictions_on_Using_Accessibility_Levels.cs
// CS0052 expected as well as CS0053, CS0056, and CS0057
// To make the program work, change access level of both class B
// and MyPrivateMethod() to public.

using System;

// A delegate:
delegate int MyDelegate();

class B
{
    // A private method:
    static int MyPrivateMethod()
    {
        return 0;
    }
}

public class A
{
    // Error: The type B is less accessible than the field A.myField.
```

```
public B myField = new B();

// Error: The type B is less accessible
// than the constant A.myConst.
public readonly B myConst = new B();

public B MyMethod()
{
    // Error: The type B is less accessible
    // than the method A.MyMethod.
    return new B();
}

// Error: The type B is less accessible than the property A.MyProp
public B MyProp
{
    set
    {
    }
}

MyDelegate d = new MyDelegate(B.MyPrivateMethod);
// Even when B is declared public, you still get the error:
// "The parameter B.MyPrivateMethod is not accessible due to
// protection level."

public static B operator +(A m1, B m2)
{
    // Error: The type B is less accessible
    // than the operator A.operator +(A,B)
    return new B();
}

static void Main()
{
    Console.WriteLine("Compiled successfully");
}
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性
- 10.2.6.5 访问包含类型的私有成员和受保护成员

请参见

参考

[C# 关键字](#)

[访问修饰符\(C# 参考\)](#)

[可访问域\(C# 参考\)](#)

[可访问性级别\(C# 参考\)](#)

[访问修饰符\(C# 编程指南\)](#)

[public\(C# 参考\)](#)

[private\(C# 参考\)](#)

[protected\(C# 参考\)](#)

[internal\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

internal(C# 参考)

internal 关键字是类型和类型成员的访问修饰符。只有在同一程序集的文件中，内部类型或成员才是可访问的，如下例所示：

```
public class BaseClass
{
    // Only accessible within the same assembly
    internal static int x = 0;
}
```

有关 **internal** 与其他访问修饰符的比较，请参见[可访问性级别和访问修饰符\(C# 编程指南\)](#)。

有关程序集的更多信息，请参见[程序集和全局程序集缓存\(C# 编程指南\)](#)：

内部访问通常用于基于组件的开发，因为它使一组组件能够以私有方式进行合作，而不必向应用程序代码的其余部分公开。例如，用于生成图形用户界面的框架可以提供“控件”类和“窗体”类，这些类通过使用具有内部访问能力的成员进行合作。由于这些成员是内部的，它们不向正在使用框架的代码公开。

从定义具有内部访问能力的类型或成员的程序集外部引用该类型或成员是错误的。

注意

尽管不能用 C# 重写 **internalvirtual** 方法，但可以用某些语言（如使用 `Ilasm.exe` 的文本 Microsoft 中间语言 (MSIL)）重写它。

示例

该示例包含两个文件：`Assembly1.cs` 和 `Assembly2.cs`。第一个文件包含内部基类 `BaseClass`。在第二个文件中，实例化 `BaseClass` 的尝试将产生错误。

```
// Assembly1.cs
// compile with: /target:library
internal class BaseClass
{
    public static int intM = 0;
}
```

```
// Assembly1_a.cs
// compile with: /reference:Assembly1.dll
class TestAccess
{
    static void Main()
    {
        BaseClass myBase = new BaseClass();    // CS0122
    }
}
```

在此示例中，使用与示例 1 中所用的文件相同的文件，并将 `BaseClass` 的可访问性级别更改为 **public**。还将成员 `intM` 的可访问性级别更改为 **internal**。在此例中，您可以实例化类，但不能访问内部成员。

```
// Assembly2.cs
// compile with: /target:library
public class BaseClass
{
    internal static int intM = 0;
}
```

```
// Assembly2_a.cs
// compile with: /reference:Assembly1.dll
public class TestAccess
{
    static void Main()
```

```
{  
    BaseClass myBase = new BaseClass(); // Ok.  
    BaseClass.intM = 444; // CS0117  
}  
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性

请参见

参考

[C# 关键字](#)

[访问修饰符 \(C# 参考\)](#)

[可访问性级别 \(C# 参考\)](#)

[修饰符 \(C# 参考\)](#)

[public \(C# 参考\)](#)

[private \(C# 参考\)](#)

[protected \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

private(C# 参考)

private 关键字是一个成员访问修饰符。私有访问是允许的最低访问级别。私有成员只有在声明它们的类和结构体中才是可访问的，如下例所示：

```
class Employee
{
    private int i;
    double d; // private access by default
}
```

同一体中的嵌套类型也可以访问那些私有成员。

在定义私有成员的类或结构外引用它会导致编译时错误。

有关 **private** 与其他访问修饰符的比较，请参见[可访问性级别](#)和[访问修饰符\(C# 编程指南\)](#)。

示例

在此示例中，Employee 类包含两个私有数据成员 name 和 salary。作为私有成员，它们只能通过成员方法来访问，因此，添加了名为 GetName 和 Salary 的公共方法，以允许对私有成员进行受控制的访问。name 成员通过公共方法来访问，salary 成员通过一个公共只读属性来访问。(有关更多信息，请参见[属性\(C# 编程指南\)](#)。)

```
// private_keyword.cs
using System;
class Employee
{
    private string name = "FirstName, LastName";
    private double salary = 100.0;

    public string GetName()
    {
        return name;
    }

    public double Salary
    {
        get { return salary; }
    }
}

class MainClass
{
    static void Main()
    {
        Employee e = new Employee();

        // The data members are inaccessible (private), so
        // then can't be accessed like this:
        //     string n = e.name;
        //     double s = e.salary;

        // 'name' is indirectly accessed via method:
        string n = e.GetName();

        // 'salary' is indirectly accessed via property
        double s = e.Salary;
    }
}
```

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 3.5.1 声明的可访问性

- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性
- 10.2.6.5 访问包含类型的私有成员和受保护成员

请参见

参考

[C# 关键字](#)

[访问修饰符\(C# 参考\)](#)

[可访问性级别\(C# 参考\)](#)

[修饰符\(C# 参考\)](#)

[public\(C# 参考\)](#)

[protected\(C# 参考\)](#)

[internal\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

protected (C# 参考)

protected 关键字是一个成员访问修饰符。受保护成员在它的类中可访问并且可由派生类访问。有关 **protected** 与其他访问修饰符的比较，请参见[可访问性级别](#)。

仅当访问通过派生类类型发生时，基类的受保护成员在派生类中才是可访问的。例如，请看以下代码段：

```
// protected_keyword.cs
using System;
class A
{
    protected int x = 123;
}

class B : A
{
    static void Main()
    {
        A a = new A();
        B b = new B();

        // Error CS1540, because x can only be accessed by
        // classes derived from A.
        // a.x = 10;

        // OK, because this class derives from A.
        b.x = 10;
    }
}
```

语句 `a.x = 10` 将生成错误，因为 `A` 不是从 `B` 派生的。

结构成员无法受保护，因为无法继承结构。

示例

在此示例中，类 `DerivedPoint` 从 `Point` 派生；因此，可以从该派生类直接访问基类的受保护成员。

```
// protected_keyword_2.cs
using System;
class Point
{
    protected int x;      protected int y;
}

class DerivedPoint: Point
{
    static void Main()
    {
        DerivedPoint dp = new DerivedPoint();

        // Direct access to protected members:
        dp.x = 10;          dp.y = 15;
        Console.WriteLine("x = {0}, y = {1}", dp.x, dp.y);
    }
}
```

输出

`x = 10, y = 15`

注释

如果将 `x` 和 `y` 的访问级别更改为 `private`，编译器将发出错误信息：

`'Point.y' is inaccessible due to its protection level.`

'Point.x' is inaccessible due to its protection level.

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.3 对实例成员的受保护访问
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符

请参见

参考

[C# 关键字](#)

[访问修饰符 \(C# 参考\)](#)

[可访问性级别 \(C# 参考\)](#)

[修饰符 \(C# 参考\)](#)

[public \(C# 参考\)](#)

[private \(C# 参考\)](#)

[internal \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

public(C# 参考)

public 关键字是类型和类型成员的访问修饰符。公共访问是允许的最高访问级别。对访问公共成员没有限制，如下例所示：

```
class SampleClass
{
    public int x; // No access restrictions.
}
```

有关更多信息，请参见[访问修饰符\(C# 编程指南\)](#)和[可访问性级别](#)。

示例

在下面的示例中，声明了两个类：`Point` 和 `MainClass`。直接从 `MainClass` 访问 `Point` 的公共成员 `x` 和 `y`。

```
// protected_public.cs
// Public access
using System;
class Point
{
    public int x;
    public int y;
}

class MainClass
{
    static void Main()
    {
        Point p = new Point();
        // Direct access to public members:
        p.x = 10;
        p.y = 15;
        Console.WriteLine("x = {0}, y = {1}", p.x, p.y);
    }
}
```

输出

`x = 10, y = 15`

如果将 **public** 访问级别更改为 [private](#) 或 [protected](#)，您将收到错误信息：

`'Point.y' is inaccessible due to its protection level.`

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 3.5.1 声明的可访问性
- 3.5.4 可访问性约束
- 10.2.3 访问修饰符
- 10.2.6.2 声明的可访问性

请参见

参考

[访问修饰符\(C# 编程指南\)](#)

[C# 关键字](#)

[访问修饰符\(C# 参考\)](#)

[可访问性级别\(C# 参考\)](#)

[修饰符\(C# 参考\)](#)

[private\(C# 参考\)](#)

[protected\(C# 参考\)](#)

[internal\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

abstract(C# 参考)

abstract 修饰符可以和类、方法、属性、索引器及事件一起使用。在类声明中使用 **abstract** 修饰符以指示某个类只能是其他类的基类。标记为抽象或包含在抽象类中的成员必须通过从抽象类派生的类来实现。

在此例中，类 `Square` 必须提供 `Area` 的实现，因为它派生自 `ShapesClass`：

```
abstract class ShapesClass
{
    abstract public int Area();
}
class Square : ShapesClass
{
    int x, y;
    // Not providing an Area method results
    // in a compile-time error.
    public override int Area()
    {
        return x * y;
    }
}
```

有关抽象类的更多信息，请参见[抽象类、密封类及类成员\(C# 编程指南\)](#)。

备注

抽象类具有以下特性：

- 抽象类不能实例化。
- 抽象类可以包含抽象方法和抽象访问器。
- 不能用 [sealed\(C# 参考\)](#) 修饰符修改抽象类，这意味着抽象类不能被继承。
- 从抽象类派生的非抽象类必须包括继承的所有抽象方法和抽象访问器的实现。

在方法或属性声明中使用 **abstract** 修饰符以指示方法或属性不包含实现。

抽象方法具有以下特性：

- 抽象方法是隐式的虚方法。
- 只允许在抽象类中使用抽象方法声明。
- 因为抽象方法声明不提供实际的实现，所以没有方法体；方法声明只是以一个分号结束，并且在签名后没有大括号 ({})。例如：

```
public abstract void MyMethod();
```

- 实现由一个重写方法[override\(C# 参考\)](#) 提供，此重写方法是非抽象类的一个成员。
- 在抽象方法声明中使用 `static` 或 `virtual` 修饰符是错误的。

除了在声明和调用语法上不同外，抽象属性的行为与抽象方法一样。

- 在静态属性上使用 **abstract** 修饰符是错误的。
- 在派生类中，通过包括使用 `override` 修饰符的属性声明，可以重写抽象的继承属性。

抽象类必须为所有接口成员提供实现。

实现接口的抽象类可以将接口方法映射到抽象方法上。例如：

```
interface I
{
```

```
    void M();
}
abstract class C: I
{
    public abstract void M();
}
```

示例

在本例中，`DerivedClass` 类是从抽象类 `BaseClass` 派生的。抽象类包含一个抽象方法 `AbstractMethod` 和两个抽象属性 `X` 和 `Y`。

```
// abstract_keyword.cs
// Abstract Classes
using System;
abstract class BaseClass // Abstract class
{
    protected int _x = 100;
    protected int _y = 150;
    public abstract void AbstractMethod(); // Abstract method
    public abstract int X { get; }
    public abstract int Y { get; }
}

class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X // overriding property
    {
        get
        {
            return _x + 10;
        }
    }

    public override int Y // overriding property
    {
        get
        {
            return _y + 10;
        }
    }

    static void Main()
    {
        DerivedClass o = new DerivedClass();
        o.AbstractMethod();
        Console.WriteLine("x = {0}, y = {1}", o.X, o.Y);
    }
}
```

输出

`x = 111, y = 161`

注释

在上面的示例中，如果试图通过使用下面的语句将抽象类实例化：

```
BaseClass bc = new BaseClass(); // Error
```

将出现错误，指出编译器无法创建抽象类“`BaseClass`”的实例。

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 1.6.5.4 虚方法、重写方法和抽象方法
- 10.1.1.1 抽象类

请参见

参考

[修饰符\(C# 参考\)](#)

[virtual\(C# 参考\)](#)

[override\(C# 参考\)](#)

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

const(C# 参考)

const 关键字用于修改字段或局部变量的声明。它指定字段或局部变量的值是常数，不能被修改。例如：

```
const int x = 0;
public const double gravitationalConstant = 6.673e-11;
private const string productName = "Visual C#";
```

备注

常数声明的类型指定声明引入的成员类型。常数表达式必须产生具有目标类型或者可隐式转换为目标类型的类型的值。

常数表达式是在编译时可被完全计算的表达式。因此，对于引用类型的常数，可能的值只能是 **string** 和 **null**。

常数声明可以声明多个常数，例如：

```
public const double x = 1.0, y = 2.0, z = 3.0;
```

不允许在常数声明中使用 **static** 修饰符。

常数可以参与常数表达式，如下所示：

```
public const int c1 = 5;
public const int c2 = c1 + 100;
```

注意

readonly 关键字与 **const** 关键字不同。**const** 字段只能在该字段的声明中初始化。**readonly** 字段可以在声明或构造函数中初始化。因此，根据所使用的构造函数，**readonly** 字段可能具有不同的值。另外，**const** 字段是编译时常量，而 **readonly** 字段可用于运行时常量，如此行所示：public static readonly uint l1 = (uint)DateTime.Now.Ticks;

示例

```
// const_keyword.cs
using System;
public class ConstTest
{
    class SampleClass
    {
        public int x;
        public int y;
        public const int c1 = 5;
        public const int c2 = c1 + 5;

        public SampleClass(int p1, int p2)
        {
            x = p1;
            y = p2;
        }
    }

    static void Main()
    {
        SampleClass mC = new SampleClass(11, 22);
        Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
        Console.WriteLine("c1 = {0}, c2 = {1}",
                         SampleClass.c1, SampleClass.c2 );
    }
}
```

输出

```
x = 11, y = 22
c1 = 5, c2 = 10
```

该示例说明如何将常数用作局部变量。

```
// const_keyword_2.cs
using System;
public class MainClass
{
    static void Main()
    {
        const int c = 707;
        Console.WriteLine("My local constant = {0}", c);
    }
}
```

输出

```
My local constant = 707
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 6.1.6 隐式常数表达式转换
- 8.5.2 局部常数声明

请参见

参考

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

[readonly\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

event(C# 参考)

event 关键字用于在发行者类中声明事件。

备注

下面的示例演示如何声明和引发将 `EventHandler` 用作基础委托类型的事件。有关演示如何使用泛型 `EventHandler<T>` 委托类型、如何订阅事件以及如何创建事件处理程序方法的完整代码示例，请参见[如何:发布符合 .NET Framework 准则的事件\(C# 编程指南\)](#)。

```
public class Publisher
{
    // Declare the delegate (if using non-generic pattern).
    public delegate void SampleEventHandler(object sender, SampleEventArgs e);

    // Declare the event.
    public event SampleEventHandler SampleEvent;

    // Wrap the event in a protected virtual method
    // to enable derived classes to raise the event.
    protected virtual void RaiseSampleEvent()
    {
        // Raise the event by using the () operator.
        SampleEvent(this, new SampleEventArgs("Hello"));
    }
}
```

事件是特殊类型的多路广播委托，仅可从声明它们的类或结构(发行者类)中调用。如果其他类或结构订阅了该事件，则当发行者类引发该事件时，会调用其事件处理程序方法。有关更多信息和代码示例，请参见[事件\(C# 编程指南\)](#)和[委托\(C# 编程指南\)](#)。

事件可标记为 `public`、`private`、`protected`、`internal` 或 `protected internal`。这些访问修饰符定义类的用户访问事件的方式。有关更多信息，请参见[访问修饰符](#)。

关键字和事件

下面的关键字可应用于事件。

关键字	说明	更多信息
<code>static</code>	即使类没有实例，调用方也能在任何时候使用该事件。	静态类和静态类成员
<code>virtual</code>	允许派生类通过使用 <code>override</code> 关键字来重写事件行为。	继承(C# 编程指南)
<code>sealed</code>	指定对于派生类它不再属虚拟性质。	
<code>abstract</code>	编译器不会生成 <code>add</code> 和 <code>remove</code> 事件访问器块，因此派生类必须提供自己的实现。	

通过使用 `static` 关键字，可以将事件声明为静态事件。即使类没有任何实例，调用方也能在任何时候使用静态事件。有关更多信息，请参见[静态类和静态类成员](#)。

通过使用 `virtual` 关键字，可以将事件标记为虚拟事件。这样，派生类就可以通过使用 `override` 关键字来重写事件行为。有关更多信息，请参见[继承\(C# 编程指南\)](#)。重写虚事件的事件也可以为 `sealed`，以此向派生类指出它不再是虚事件。最后，可以将事件声明为 `abstract`，这意味着编译器不会生成 `add` 和 `remove` 事件访问器块，因此派生类必须提供自己的实现。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 1.6.6.4 事件
- 7.13.3 事件分配
- 10.7 事件

- 13.2.3 接口事件

[请参见](#)

[任务](#)

[如何:合并委托\(多路广播委托\)\(C# 编程指南\)](#)

[参考](#)

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

extern(C# 参考)

extern 修饰符用于声明在外部实现的方法。**extern** 修饰符的常见用法是在使用 Interop 服务调入非托管代码时与 **DllImport** 属性一起使用；在这种情况下，该方法还必须声明为 **static**，如下面的示例所示：

```
[DllImport("avifil32.dll")]
private static extern void AVIFileInit();
```

注意

extern 关键字还可以定义外部程序集别名，使得可以从单个程序集中引用同一组件的不同版本。有关更多信息，请参见 [外部别名\(C# 参考\)](#)。

将 [abstract\(C# 参考\)](#) 和 **extern** 修饰符一起使用来修改同一成员是错误的。使用 **extern** 修饰符意味着方法在 C# 代码的外部实现，而使用 **abstract** 修饰符意味着在类中未提供方法实现。

注意

extern 关键字在使用上比在 C++ 中有更多的限制。若要与 C++ 关键字进行比较，请参见 C++ Language Reference 中的 [Using extern to Specify Linkage](#)。

示例

在该示例中，程序接收来自用户的字符串并将该字符串显示在消息框中。程序使用从 User32.dll 库导入的 **MessageBox** 方法。

```
using System;
using System.Runtime.InteropServices;
class MainClass
{
    [DllImport("User32.dll")]
    public static extern int MessageBox(int h, string m, string c, int type);

    static int Main()
    {
        string myString;
        Console.Write("Enter your message: ");
        myString = Console.ReadLine();
        return MessageBox(0, myString, "My Message Box", 0);
    }
}
```

此示例使用 C 程序创建一个 DLL，在下一示例中将从 C# 程序调用该 DLL。

```
// cmdll.c
// compile with: /LD
int __declspec(dllexport) SampleMethod(int i)
{
    return i*10;
}
```

该示例使用两个文件 CM.cs 和 Cmdll.c 来说明 **extern**。C 文件是示例 2 中创建的外部 DLL，它从 C# 程序内调用。

```
// cm.cs
using System;
using System.Runtime.InteropServices;
public class MainClass
{
    [DllImport("Cmdll.dll")]
    public static extern int SampleMethod(int x);

    static void Main()
```

```
    {
        Console.WriteLine("SampleMethod() returns {0}.", SampleMethod(5));
    }
}
```

输出

SampleMethod() returns 50.

备注

生成项目：

- 使用 Visual C++ 命令行将 Cmdll.c 编译为 DLL：

```
cl /LD Cmdll.c
```

- 使用命令行编译 CM.cs：

```
csc CM.cs
```

这将创建可执行文件 CM.exe。运行此程序时，SampleMethod 将值 5 传递到 DLL 文件，该文件将此值乘以 10 返回。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.5.7 外部方法

请参见

参考

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

[System.Runtime.InteropServices.DllImportAttribute](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

override(C# 参考)

要扩展或修改继承的方法、属性、索引器或事件的抽象实现或虚实现，必须使用 **override** 修饰符。

在此例中，类 `Square` 必须提供 `Area` 的重写实现，因为 `Area` 是从抽象的 `ShapesClass` 继承而来的。

```
abstract class ShapesClass
{
    abstract public int Area();
}

class Square : ShapesClass
{
    int x, y;
    // Because ShapesClass.Area is abstract, failing to override
    // the Area method would result in a compilation error.
    public override int Area()
    {
        return x * y;
    }
}
```

有关 **override** 关键字用法的更多信息，请参见[使用 Override 和 New 关键字进行版本控制](#)以及[了解何时使用 Override 和 New 关键字](#)。

备注

override 方法提供从基类继承的成员的新实现。通过 **override** 声明重写的方法称为重写基方法。重写的基方法必须与 **override** 方法具有相同的签名。有关继承的信息，请参见[继承](#)。

不能重写非虚方法或静态方法。重写的基方法必须是 **virtual**、**abstract** 或 **override** 的。

override 声明不能更改 **virtual** 方法的可访问性。**override** 方法和 **virtual** 方法必须具有相同的[访问级别修饰符](#)。

不能使用修饰符 **new**、**static**、**virtual** 或 **abstract** 来修改 **override** 方法。

重写属性声明必须指定与继承属性完全相同的访问修饰符、类型和名称，并且被重写的属性必须是 **virtual**、**abstract** 或 **override** 的。

示例

此示例定义了一个名为 `Employee` 的基类，和一个名为 `SalesEmployee` 的派生类。`SalesEmployee` 类包括一个额外的属性 `salesbonus`，并重写方法 `CalculatePay` 以便将该属性考虑在内。

```
using System;
class TestOverride
{
    public class Employee
    {
        public string name;

        // Basepay is defined as protected, so that it may be
        // accessed only by this class and derived classes.
        protected decimal basepay;

        // Constructor to set the name and basepay values.
        public Employee(string name, decimal basepay)
        {
            this.name = name;
            this.basepay = basepay;
        }

        // Declared virtual so it can be overridden.
        public virtual decimal CalculatePay()
        {
            return basepay;
        }
    }
}
```

```

    }

// Derive a new class from Employee.
public class SalesEmployee : Employee
{
    // New field that will affect the base pay.
    private decimal salesbonus;

    // The constructor calls the base-class version, and
    // initializes the salesbonus field.
    public SalesEmployee(string name, decimal basepay,
                         decimal salesbonus) : base(name, basepay)
    {
        this.salesbonus = salesbonus;
    }

    // Override the CalculatePay method
    // to take bonus into account.
    public override decimal CalculatePay()
    {
        return basepay + salesbonus;
    }
}

static void Main()
{
    // Create some new employees.
    SalesEmployee employee1 = new SalesEmployee("Alice",
                                                1000, 500);
    Employee employee2 = new Employee("Bob", 1200);

    Console.WriteLine("Employee " + employee1.name +
                      " earned: " + employee1.CalculatePay());
    Console.WriteLine("Employee " + employee2.name +
                      " earned: " + employee2.CalculatePay());
}
}

```

输出

Employee Alice earned: 1500
 Employee Bob earned: 1200

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.5.4 虚方法、重写方法和抽象方法
- 10.5.4 重写方法

请参见

参考

[继承 \(C# 编程指南\)](#)

[C# 关键字](#)

[修饰符 \(C# 参考\)](#)

[abstract \(C# 参考\)](#)

[virtual \(C# 参考\)](#)

[new \(C# 参考\)](#)

概念

[C# 编程指南](#)

[多态性 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

readonly(C# 参考)

readonly 关键字是可以在字段上使用的修饰符。当字段声明包括 **readonly** 修饰符时，该声明引入的字段赋值只能作为声明的一部分出现，或者出现在同一类的构造函数中。在此示例中，字段 `year` 的值无法在 `ChangeYear` 方法中更改，即使在类构造函数中给它赋了值。

```
class Age
{
    readonly int _year;
    Age(int year)
    {
        _year = year;
    }
    void ChangeYear()
    {
        _year = 1967; // Will not compile.
    }
}
```

备注

只能在下列上下文中对 **readonly** 字段进行赋值：

- 当在声明中初始化变量时，例如：

```
public readonly int y = 5;
```

- 对于实例字段，在包含字段声明的类的实例构造函数中；或者，对于静态字段，在包含字段声明的类的静态构造函数中。也只有在这些上下文中，将 `readonly` 字段作为 `out` 或 `ref` 参数传递才有效。

注意

readonly 关键字与 `const` 关键字不同。`const` 字段只能在该字段的声明中初始化。**readonly** 字段可以在声明或构造函数中初始化。因此，根据所使用的构造函数，**readonly** 字段可能具有不同的值。另外，`const` 字段为编译时常数，而 **readonly** 字段可用于运行时常数，如下例所示：

注意

```
public static readonly uint l1 = (uint)DateTime.Now.Ticks;
```

示例

```
// cs_READONLY_keyword.cs
// Readonly fields
using System;
public class ReadOnlyTest
{
    class SampleClass
    {
        public int x;
        // Initialize a readonly field
        public readonly int y = 25;
        public readonly int z;

        public SampleClass()
        {
            // Initialize a readonly instance field
            z = 24;
        }

        public SampleClass(int p1, int p2, int p3)
        {
            x = p1;
```

```
        y = p2;
        z = p3;
    }

    static void Main()
{
    SampleClass p1 = new SampleClass(11, 21, 32); // OK
    Console.WriteLine("p1: x={0}, y={1}, z={2}", p1.x, p1.y, p1.z);
    SampleClass p2 = new SampleClass();
    p2.x = 55; // OK
    Console.WriteLine("p2: x={0}, y={1}, z={2}", p2.x, p2.y, p2.z);
}
}
```

输出

```
p1: x=11, y=21, z=32
p2: x=55, y=25, z=24
```

注释

在前面的示例中，如果使用这样的语句：

```
p2.y = 66; // Error
```

将收到编译器错误信息：

```
The left-hand side of an assignment must be an l-value
```

这与试图将值赋给常数时收到的错误相同。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.4.2 只读字段

请参见

参考

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

[const\(C# 参考\)](#)

[字段\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

sealed(C# 参考)

sealed 修饰符可以应用于类、实例方法和属性。密封类不能被继承。密封方法会重写基类中的方法，但其本身不能在任何派生类中进一步重写。当应用于方法或属性时，**sealed** 修饰符必须始终与 [override\(C# 参考\)](#) 一起使用。

在类声明中使用 **sealed** 修饰符可防止继承此类，例如：

```
sealed class SealedClass
{
    public int x;
    public int y;
}
```

将密封类用作基类或将 **abstract** 修饰符与密封类一起使用是错误的。

结构是隐式密封的；因此它们不能被继承。

有关继承的更多信息，请参见[继承\(C# 编程指南\)](#)。

示例

```
// cs_sealed_keyword.cs
using System;
sealed class SealedClass
{
    public int x;
    public int y;
}

class MainClass
{
    static void Main()
    {
        SealedClass sc = new SealedClass();
        sc.x = 110;
        sc.y = 150;
        Console.WriteLine("x = {0}, y = {1}", sc.x, sc.y);
    }
}
```

输出

x = 110, y = 150

在前面的示例中，如果试图通过使用下面的语句从密封类继承：

```
class MyDerivedC: SealedClass {} // Error
```

将收到错误消息：

```
'MyDerivedC' cannot inherit from sealed class 'SealedClass'.
```

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 10.1.1.2 密封类
- 10.5.5 密封方法

请参见

参考

C# 关键字

[静态类和静态类成员\(C# 编程指南\)](#)

[抽象类、密封类及类成员\(C# 编程指南\)](#)

[访问修饰符\(C# 编程指南\)](#)

[修饰符\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

static(C# 参考)

使用 **static** 修饰符声明属于类型本身而不是属于特定对象的静态成员。**static** 修饰符可用于类、字段、方法、属性、运算符、事件和构造函数，但不能用于索引器、析构函数或类以外的类型。例如，下面的类声明为 **static**，并且只包含 **static** 方法。

```
static class CompanyEmployee
{
    public static string GetCompanyName(string name) { ... }
    public static string GetCompanyAddress(string address) { ... }
}
```

有关更多信息，请参见[静态类和静态类成员\(C# 编程指南\)](#)。

备注

- 常数或者类型声明隐式地是静态成员。
- 不能通过实例引用静态成员。然而，可以通过类型名称引用它。例如，请考虑以下类：

```
public class MyBaseC
{
    public struct MyStruct
    {
        public static int x = 100;
    }
}
```

若要引用静态成员 `x`，请使用完全限定名(除非可从相同范围访问)：

```
MyBaseC.MyStruct.x
```

- 尽管类的实例包含该类所有实例字段的单独副本，但每个静态字段只有一个副本。
- 不可以使用 `this` 来引用静态方法或属性访问器。
- 如果对类应用 **static** 关键字，则该类的所有成员都必须是静态的。
- 类(包括静态类)可以有静态构造函数。在程序开始和实例化类之间的某个时刻调用静态构造函数。

注意

static 关键字在使用上比在 C++ 中有更多的限制。若要与 C++ 关键字比较，请参见 [static](#)。

为了说明静态成员，请看一个表示公司雇员的类。假设该类包含一种对雇员计数的方法和一个存储雇员数的字段。该方法和字段都不属于任何实例雇员，而是属于公司类。因此，应该将它们声明为此类的静态成员。

示例

该示例读取新雇员的名称和 ID，逐个增加雇员计数器并显示新雇员的有关信息以及新的雇员数。为简单起见，该程序从键盘读取当前的雇员数。在实际的应用中，应从文件读取此信息。

```
// cs_static_keyword.cs
using System;

public class Employee
{
    public string id;
    public string name;

    public Employee()
    {
        static int count = 0;
        count++;
        Console.WriteLine("Employee " + count);
        Console.WriteLine("Name: " + name);
        Console.WriteLine("ID: " + id);
    }
}
```

```

}

public Employee(string name, string id)
{
    this.name = name;
    this.id = id;
}

public static int employeeCounter;

public static int AddEmployee()
{
    return ++employeeCounter;
}

class MainClass : Employee
{
    static void Main()
    {
        Console.WriteLine("Enter the employee's name: ");
        string name = Console.ReadLine();
        Console.WriteLine("Enter the employee's ID: ");
        string id = Console.ReadLine();

        // Create and configure the employee object:
        Employee e = new Employee(name, id);
        Console.WriteLine("Enter the current number of employees: ");
        string n = Console.ReadLine();
        Employee.employeeCounter = Int32.Parse(n);
        Employee.AddEmployee();

        // Display the new information:
        Console.WriteLine("Name: {0}", e.name);
        Console.WriteLine("ID: {0}", e.id);
        Console.WriteLine("New Number of Employees: {0}",
                         Employee.employeeCounter);
    }
}

```

输入

Tara Strahan
AF643G
15

示例输出

```

Enter the employee's name: Tara Strahan
Enter the employee's ID: AF643G
Enter the current number of employees: 15
Name: Tara Strahan
ID: AF643G
New Number of Employees: 16

```

此示例显示：虽然可以用另一个尚未声明的静态字段实例化一个静态字段，但只要给静态字段显式赋值后，结果才是确定的。

```

// cs_static_keyword_2.cs
using System;
class Test
{
    static int x = y;
    static int y = 5;

    static void Main()
    {
        Console.WriteLine(Test.x);
    }
}

```

```
        Console.WriteLine(Test.y);

        Test.x = 99;
        Console.WriteLine(Test.x);
    }
}
```

输出

```
0
5
99
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.5.3 静态方法和实例方法
- 5.1.1 静态变量
- 10.2.5 静态成员和实例成员
- 10.4.1 静态字段和实例字段
- 10.4.5.1 静态字段初始化
- 10.5.2 静态方法和实例方法
- 10.6.1 静态属性和实例属性
- 10.7.3 静态事件和实例事件
- 10.11 静态构造函数

请参见

参考

[C# 关键字](#)

[修饰符 \(C# 参考\)](#)

[静态类和静态类成员 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

unsafe(C# 参考)

unsafe 关键字表示不安全上下文，该上下文是任何涉及指针的操作所必需的。有关更多信息，请参见[不安全代码和指针\(C# 编程指南\)](#)。

可以在类型或成员的声明中使用 **unsafe** 修饰符。因此，类型或成员的整个正文范围均被视为不安全上下文。例如，以下是用 **unsafe** 修饰符声明的方法：

```
unsafe static void FastCopy(byte[] src, byte[] dst, int count)
{
    // Unsafe context: can use pointers here.
}
```

不安全上下文的范围从参数列表扩展到方法的结尾，因此指针在以下参数列表中也可以使用：

```
unsafe static void FastCopy ( byte* ps, byte* pd, int count ) {...}
```

还可以使用不安全块从而能够使用该块内的不安全代码。例如：

```
unsafe
{
    // Unsafe context: can use pointers here.
}
```

若要编译不安全代码，必须指定[/unsafe](#) 编译器选项。无法通过公共语言运行库验证不安全代码。

示例

```
// cs_unsafe_keyword.cs
// compile with: /unsafe
using System;
class UnsafeTest
{
    // Unsafe method: takes pointer to int:
    unsafe static void SquarePtrParam(int* p)
    {
        *p *= *p;
    }

    unsafe static void Main()
    {
        int i = 5;
        // Unsafe method: uses address-of operator (&):
        SquarePtrParam(&i);
        Console.WriteLine(i);
    }
}
```

输出

25

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 18 不安全代码

请参见 参考

[C# 关键字](#)

[fixed 语句\(C# 参考\)](#)

[固定大小的缓冲区\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

virtual(C# 参考)

virtual 关键字用于修饰方法、属性、索引器或事件声明，并且允许在派生类中重写这些对象。例如，此方法可被任何继承它的类重写。

```
public virtual double Area()
{
    return x * y;
}
```

虚拟成员的实现可由派生类中的**重写成员**更改。有关使用 **virtual** 关键字的更多信息，请参见[使用 Override 和 New 关键字进行版本控制\(C# 编程指南\)](#)和[了解何时使用 Override 和 New 关键字\(C# 编程指南\)](#)。

备注

调用虚方法时，将为重写成员检查该对象的运行时类型。将调用大部分派生类中的该重写成员，如果没有派生类重写该成员，则它可能是原始成员。

默认情况下，方法是非虚拟的。不能重写非虚方法。

virtual 修饰符不能与 **static**、**abstract**、**private** 或 **override** 修饰符一起使用。

除了声明和调用语法不同外，虚拟属性的行为与抽象方法一样。

- 在静态属性上使用 **virtual** 修饰符是错误的。
- 通过包括使用 **override** 修饰符的属性声明，可在派生类中重写虚拟继承属性。

示例

在该示例中，`Dimensions` 类包含 `x` 和 `y` 两个坐标和 `Area()` 虚方法。不同的形状类，如 `Circle`、`Cylinder` 和 `Sphere` 继承 `Dimensions` 类，并为每个图形计算表面积。每个派生类都有各自的 `Area()` 重写实现。根据与此方法关联的对象，通过调用正确的 `Area()` 实现，该程序为每个图形计算并显示正确的面积。

在前面的示例中，注意继承的类 `Circle`、`Sphere` 和 `Cylinder` 都使用了初始化基类的构造函数，例如：

```
public Cylinder(double r, double h): base(r, h) {}
```

这类似于 C++ 的初始化列表。

```
// cs_virtual_keyword.cs
using System;
class TestClass
{
    public class Dimensions
    {
        public const double PI = Math.PI;
        protected double x, y;
        public Dimensions()
        {
        }
        public Dimensions(double x, double y)
        {
            this.x = x;
            this.y = y;
        }

        public virtual double Area()
        {
            return x * y;
        }
    }

    public class Circle : Dimensions
    {
```

```

public Circle(double r) : base(r, 0)
{
}

public override double Area()
{
    return PI * x * x;
}
}

class Sphere : Dimensions
{
    public Sphere(double r) : base(r, 0)
    {
    }

    public override double Area()
    {
        return 4 * PI * x * x;
    }
}

class Cylinder : Dimensions
{
    public Cylinder(double r, double h) : base(r, h)
    {
    }

    public override double Area()
    {
        return 2 * PI * x * x + 2 * PI * x * y;
    }
}

static void Main()
{
    double r = 3.0, h = 5.0;
    Dimensions c = new Circle(r);
    Dimensions s = new Sphere(r);
    Dimensions l = new Cylinder(r, h);
    // Display results:
    Console.WriteLine("Area of Circle    = {0:F2}", c.Area());
    Console.WriteLine("Area of Sphere    = {0:F2}", s.Area());
    Console.WriteLine("Area of Cylinder = {0:F2}", l.Area());
}
}

```

输出

```

Area of Circle    = 28.27
Area of Sphere    = 113.10
Area of Cylinder = 150.80

```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.5.4 虚方法、重写方法和抽象方法
- 10.5.3 虚方法
- 10.6.3 虚访问器、密封访问器、重写访问器和抽象访问器

请参见

参考

[修饰符\(C# 参考\)](#)

[C# 关键字](#)

[abstract\(C# 参考\)](#)

[override\(C# 参考\)](#)

[new\(C# 参考\)](#)

概念

[C# 编程指南](#)

[多态性\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

volatile(C# 参考)

volatile 关键字表示字段可能被多个并发执行线程修改。声明为 **volatile** 的字段不受编译器优化(假定由单个线程访问)的限制。这样可以确保该字段在任何时间呈现的都是最新的值。

volatile 修饰符通常用于由多个线程访问而不使用 [lock 语句\(C# 参考\)](#) 语句对访问进行序列化的字段。有关在多线程方案中使用 **volatile** 的示例, 请参见[如何: 创建和终止线程\(C# 编程指南\)](#)。

volatile 关键字可应用于以下类型的字段:

- 引用类型。
- 指针类型(在不安全的上下文中)。请注意, 虽然指针本身可以是可变的, 但是它指向的对象不能是可变的。换句话说, 您无法声明“指向可变对象的指针”。
- 整型, 如 sbyte、byte、short、ushort、int、uint、char、float 和 bool。
- 具有整数基类型的枚举类型。
- 已知为引用类型的泛型类型参数。
- [IntPtr](#) 和 [UIntPtr](#)。

可变关键字仅可应用于类或结构字段。不能将局部变量声明为 **volatile**。

示例

下面的示例说明如何将公共字段变量声明为 **volatile**。

```
// csharp_volatile.cs
// compile with: /target:library
class Test
{
    public volatile int i;

    Test(int _i)
    {
        i = _i;
    }
}
```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 3.10 执行顺序
- 10.4.3 可变字段

请参见

参考

[C# 关键字](#)

[修饰符\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

语句类型(C# 参考)

语句是程序指令。除非特别说明，语句都按顺序执行。C# 具有下列类别的语句。

类别	C# 关键字
选择语句	if , else , switch , case
迭代语句	do , for , foreach , in , while
跳转语句	break , continue , default , goto , return , yield
异常处理语句	throw , try-catch , try-finally , try-catch-finally
检查和未检查	checked , unchecked
fixed 语句	fixed
lock 语句	lock

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

选择语句(C# 参考)

选择语句根据某个条件是否为 **true** 来将程序控制权移交给特定的流。

选择语句中使用下列关键字：

- [if](#)
- [else](#)
- [switch](#)
- [case](#)
- [default](#)

请参见

参考

[C# 关键字](#)

[语句类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

if-else(C# 参考)

if 语句根据 **Boolean** 表达式的值选择要执行的语句。下面的示例将 **Boolean** 标志 `flagCheck` 设置为 **true**, 然后在 **if** 语句中检查该标志。输出为: `The flag is set to true.`

```
bool flagCheck = true;
if (flagCheck == true)
{
    Console.WriteLine("The flag is set to true.");
}
else
{
    Console.WriteLine("The flag is set to false.");
}
```

备注

如果括号里的表达式计算为 **true**, 则执行 `Console.WriteLine("The boolean flag is set to ture.");` 语句。执行 **if** 语句之后, 控制传递给下一个语句。在此例中不执行 **else** 语句。

如果想要执行的语句不止一个, 可以通过使用 {} 将多个语句包含在块中, 有条件地执行多个语句, 如上例所示。

在测试条件时执行的语句可以是任何种类的, 包括嵌套在原始 **if** 语句中的另一个 **if** 语句。在嵌套的 **if** 语句中, **else** 子句属于最后一个没有对应的 **else** 的 **if**。例如:

```
if (x > 10)
if (y > 20)
    Console.Write("Statement_1");
else
    Console.Write("Statement_2");
```

在此例中, 如果条件 `(y > 20)` 计算为 **false**, 将显示 `Statement_2`。但如果要使 `Statement_2` 与条件 `(x > 10)` 关联, 则使用大括号:

```
{           if (x > 10)
if (y > 20)
    Console.Write("Statement_1");
}
else
    Console.Write("Statement_2");
```

在此例中, 如果条件 `(x > 10)` 计算为 **false**, 将显示 `Statement_2`

示例 1

在此例中, 您从键盘输入一个字符, 而程序检查输入字符是否为字母字符。如果输入的字符是字母, 则程序检查是大写还是小写。在任何一种情况下, 都会显示适当的消息。

```
// statements_if_else.cs
// if-else example
using System;
class IfTest
{
    static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();
        if (Char.IsLetter(c))
        {
            if (Char.ToLower(c))
```

```
        {
            Console.WriteLine("The character is lowercase.");
        }
        else
        {
            Console.WriteLine("The character is uppercase.");
        }
    }
    else
    {
        Console.WriteLine("Not an alphabetic character.");
    }
}
```

输入

2

示例输出

```
Enter a character: 2
The character is not an alphabetic character.
```

附加的示例可能如下所示：

运行示例 #2：

```
Enter a character: A
The character is uppercase.
```

运行示例 #3：

```
Enter a character: h
The character is lowercase.
```

还可以扩展 **if** 语句，使用下列 **else-if** 排列来处理多个条件：

```
if (Condition_1)
{
    // Statement_1;
}
else if (Condition_2)
{
    // Statement_2;
}
else if (Condition_3)
{
    // Statement_3;
}
else
{
    // Statement_n;
}
```

示例 2

此示例检查输入字符是否是小写字符、大写字符或数字。否则，输入字符不是字母字符。程序利用 **else-if** 阶梯。

```
// statements_if_else2.cs
// else-if
using System;
public class IfTest
{
    static void Main()
    {
        Console.Write("Enter a character: ");
        char c = (char)Console.Read();

        if (Char.IsUpper(c))

```

```
{  
    Console.WriteLine("Character is uppercase.");  
}  
else if (Char.IsLower(c))  
{  
    Console.WriteLine("Character is lowercase.");  
}  
else if (Char.IsDigit(c))  
{  
    Console.WriteLine("Character is a number.");  
}  
else  
{  
    Console.WriteLine("Character is not alphanumeric.");  
}  
}  
}
```

输入

E

示例输出

```
Enter a character: E  
The character is uppercase.
```

附加的运行示例可能如下所示：

运行示例 #2：

```
Enter a character: e  
The character is lowercase.
```

运行示例 #3：

```
Enter a character: 4  
The character is a number.
```

运行示例 #4：

```
Enter a character: $  
The character is not alphanumeric.
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.5 If 语句
- 8.7.1 if 语句

请参见

参考

[C# 关键字](#)

[?: 运算符\(C# 参考\)](#)

[The if-else Statement](#)

[switch\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

switch(C# 参考)

switch 语句是一个控制语句，它通过将控制传递给其体内的一一个 **case** 语句来处理多个选择和枚举。例如：

```
int caseSwitch = 1;
switch (caseSwitch)
{
    case 1:
        Console.WriteLine("Case 1");
        break;
    case 2:
        Console.WriteLine("Case 2");
        break;
    default:
        Console.WriteLine("Default case");
        break;
}
```

备注

控制传递给与开关的值匹配的 **case** 语句。**switch** 语句可以包括任意数目的 **case** 实例，但是任何两个 **case** 语句都不能具有相同的值。语句体从选定的语句开始执行，直到 **break** 将控制传递到 **case** 体以外。在每一个 **case** 块（包括上一个块，不论它是 **case** 语句还是 **default** 语句）的后面，都必须有一个跳转语句（如 **break**）。但有一个例外，（与 C++ **switch** 语句不同）C# 不支持从一个 **case** 标签显式贯穿到另一个 **case** 标签。这个例外是当 **case** 语句中没有代码时。

如果没有任何 **case** 表达式与开关值匹配，则控制传递给跟在可选 **default** 标签后的语句。如果没有 **default** 标签，则控制传递到 **switch** 以外。

示例

```
// statements_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch(n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
                cost += 25;
                goto case 1;
            case 3:
                cost += 50;
                goto case 1;
            default:
                Console.WriteLine("Invalid selection. Please select 1, 2, or 3.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
```

输入

示例输出

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

下面的示例显示了空 case 标签可以从一个 case 标签贯穿到另一个。

```
// statements_switch2.cs
using System;
class SwitchTest
{
    static void Main()
    {
        int n = 2;
        switch(n)
        {
            case 1:
            case 2:
            case 3:
                Console.WriteLine("It's 1, 2, or 3.");
                break;
            default:
                Console.WriteLine("Not sure what it is.");
                break;
        }
    }
}
```

输出

It's 1, 2, or 3.

代码讨论

- 在前面的示例中，整型变量 `n` 用于 `switch case`。注意还可以直接使用字符串变量 `s`。在这种情况下，可以以下列方式使用 `switch case`：

```
switch(s)
{
    case "1":
        // ...
    case "2":
        // ...
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [5.3.3.6 switch 语句](#)
- [8.7.2 switch 语句](#)

请参见

参考

[C# 关键字](#)

[The switch Statement](#)

[if-else\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

迭代语句(C# 参考)

通过使用迭代语句可以创建循环。迭代语句导致嵌入语句根据循环终止条件多次执行。除非遇到[跳转语句](#)，否则这些语句将按顺序执行。

迭代语句中使用下列关键字：

- [do](#)
- [for](#)
- [foreach](#)
- [in](#)
- [while](#)

请参见

参考

[C# 关键字](#)

[语句类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

do(C# 参考)

do 语句重复执行括在 {} 里的一个语句或语句块，直到指定的表达式计算为 **false**。在下面的示例中，只要变量 *y* 小于 5，**do-while** 循环语句就开始执行。

示例

```
// statements_do.cs
using System;
public class TestDoWhile
{
    public static void Main ()
    {
        int x = 0;
        do
        {
            Console.WriteLine(x);
            x++;
        } while (x < 5);
    }
}
```

输出

```
0
1
2
3
4
```

备注

与 **while** 语句不同的是，**do-while** 循环会在计算条件表达式之前执行一次。

在 **do-while** 块中的任何点，都可用使用 **break** 语句跳出循环。使用 **continue** 语句可以直接进入 **while** 表达式计算语句；如果表达式计算结果为 **true**，则会继续从循环的第一个语句执行。如果表达式计算结果为 **false**，则会继续从 **do-while** 循环后的第一个语句执行。

do-while 循环还可以通过 **goto**、**return** 或 **throw** 语句退出。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [5.3.3.8 do 语句](#)
- [8.8.2 do 语句](#)

请参见

参考

[C# 关键字](#)

[The do-while Statement \(C++\)](#)

[迭代语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

for(C# 参考)

for 循环重复执行一个语句或语句块，直到指定的表达式计算为 **false** 值。**for** 循环对于迭代数组和顺序处理非常方便。在下面的示例中，`int i` 的值写入控制台，并且 `i` 在每次通过循环时都加 1。

示例

```
// statements_for.cs
// for loop
using System;
class ForLoopTest
{
    static void Main()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine(i);
        }
    }
}
```

输出

```
1
2
3
4
5
```

备注

for 语句重复执行括起来的语句，如下所述：

- 首先，计算变量 `i` 的初始值。
- 当 `i` 的值小于或等于 5 时，条件计算结果为 **true**，将执行 `Console.WriteLine` 语句并会重新计算 `i`。
- 当 `i` 大于 5 时，条件变成 **false** 并且控制传递到循环外部。

由于条件表达式的测试在循环执行之前发生，因此 **for** 语句执行零次或更多次。

for 语句的所有表达式都是可选的；例如，下列语句用于写一个无限循环：

```
for (;;)
{
    // ...
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.9 `for` 语句
- 8.8.3 `for` 语句

请参见

参考

[C# 关键字](#)

[foreach, in\(C# 参考\)](#)

[The for Statement](#)

[迭代语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

foreach, in(C# 参考)

foreach 语句为数组或对象集合中的每个元素重复一个嵌入语句组。**foreach** 语句用于循环访问集合以获取所需信息，但不应用于更改集合内容以避免产生不可预知的副作用。

备注

嵌入语句为数组或集合中的每个元素继续执行。当为集合中的所有元素完成迭代后，控制传递给 **foreach** 块之后的下一个语句。

可以在 **foreach** 块的任何点使用 [break](#) 关键字跳出循环，或使用 [continue](#) 关键字直接进入循环的下一轮迭代。

foreach 循环还可以通过 [goto](#)、[return](#) 或 [throw](#) 语句退出。

有关 **foreach** 关键字的更多信息(包括代码示例)，请参见下面的主题：

[对数组使用 foreach\(C# 编程指南\)](#)

[如何: 使用 foreach 访问集合类\(C# 编程指南\)](#)

示例

在此示例中，使用 **foreach** 显示整数数组的内容。

```
// cs.foreach.cs
class ForEachTest
{
    static void Main(string[] args)
    {
        int[] fibarray = new int[] { 0, 1, 2, 3, 5, 8, 13 };
        foreach (int i in fibarray)
        {
            System.Console.WriteLine(i);
        }
    }
}
```

输出

```
0
1
2
3
5
8
13
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.16 foreach 语句
- 8.8.4 foreach 语句

请参见

参考

[C# 关键字](#)

[迭代语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

while(C# 参考)

while 语句执行一个语句或语句块，直到指定的表达式计算为 **false**。

示例

```
// statements_while.cs
using System;
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
            n++;
        }
    }
}
```

输出

```
Current value of n is 1
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
```

```
// statements_while_2.cs
using System;
class WhileTest
{
    static void Main()
    {
        int n = 1;
        while (n++ < 6)
        {
            Console.WriteLine("Current value of n is {0}", n);
        }
    }
}
```

输出

```
Current value of n is 2
Current value of n is 3
Current value of n is 4
Current value of n is 5
Current value of n is 6
```

由于 **while** 表达式的测试在每次执行循环前发生，因此 **while** 循环执行零次或更多次。这与执行一次或多次的 **do** 循环不同。

当 [break](#)、[goto](#)、[return](#) 或 [throw](#) 语句将控制权转移到 **while** 循环之外时，可以终止该循环。若要将控制权传递给下一次迭代但不退出循环，请使用 [continue](#) 语句。请注意，在上面三个示例中，根据 `int n` 递增的位置的不同，输出也不同。在下面的示例中不生成输出。

```
// statements_while_3.cs
// no output is generated
using System;
class WhileTest
{
    static void Main()
    {
        int n = 5;
```

```
    while (++n < 6)
    {
        Console.WriteLine("Current value of n is {0}", n);
    }
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.7 While 语句
- 8.8.1 while 语句

请参见

参考

[C# 关键字](#)

[The while Statement](#)

[迭代语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

跳转语句(C# 参考)

使用跳转语句执行分支，该语句导致立即传递程序控制。跳转语句中使用下列关键字：

- [break](#)
- [continue](#)
- [goto](#)
- [return](#)
- [throw](#)

请参见

参考

[C# 关键字](#)

[语句类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

break(C# 参考)

break 语句用于终止最近的封闭循环或它所在的 [switch](#) 语句。控制传递给终止语句后面的语句(如果有的话)。

示例

在此例中, 条件语句包含一个应该从 1 计数到 100 的计数器;但 **break** 语句在计数达到 4 后终止循环。

```
// statements_break.cs
using System;
class BreakTest
{
    static void Main()
    {
        for (int i = 1; i <= 100; i++)
        {
            if (i == 5)
            {
                break;
            }
            Console.WriteLine(i);
        }
    }
}
```

输出

```
1
2
3
4
```

下面的示例演示 **break** 在 [switch](#) 语句中的用法。

```
// statements_break2.cs
// break and switch
using System;
class Switch
{
    static void Main()
    {
        Console.Write("Enter your selection (1, 2, or 3): ");
        string s = Console.ReadLine();
        int n = Int32.Parse(s);

        switch (n)
        {
            case 1:
                Console.WriteLine("Current value is {0}", 1);
                break;
            case 2:
                Console.WriteLine("Current value is {0}", 2);
                break;
            case 3:
                Console.WriteLine("Current value is {0}", 3);
                break;
            default:
                Console.WriteLine("Sorry, invalid selection.");
                break;
        }
    }
}
```

输入

示例输出

```
Enter your selection (1, 2, or 3): 1
Current value is 1
```

注释

如果输入了 4，则输出为：

```
Enter your selection (1, 2, or 3): 4
Sorry, invalid selection.
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.10 Break、continue 和 goto 语句
- 8.9.1 break 语句

请参见

参考

[C# 关键字](#)

[The break Statement](#)

[switch\(C# 参考\)](#)

[跳转语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

continue(C# 参考)

continue 语句将控制权传递给它所在的封闭迭代语句的下一次迭代。

示例

在此示例中，计数器最初是从 1 到 10 进行计数，但通过将 **continue** 语句与表达式 (*i* < 9) 一起使用，跳过了 **continue** 与 **for** 循环体末尾之间的语句。

```
// statements_continue.cs
using System;
class ContinueTest
{
    static void Main()
    {
        for (int i = 1; i <= 10; i++)
        {
            if (i < 9)
            {
                continue;
            }
            Console.WriteLine(i);
        }
    }
}
```

输出

```
9
10
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.10 Break、continue 和 goto 语句
- 8.9.2 continue 语句

请参见

参考

[C# 关键字](#)

[The break Statement](#)

[跳转语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

goto(C# 参考)

goto 语句将程序控制直接传递给标记语句。

备注

goto 的一个通常用法是将控制传递给特定的 switch-case 标签或 **switch** 语句中的默认标签。

goto 语句还用于跳出深嵌套循环。

示例

下面的示例演示了 **goto** 在 **switch** 语句中的使用。

```
// statements_goto_switch.cs
using System;
class SwitchTest
{
    static void Main()
    {
        Console.WriteLine("Coffee sizes: 1=Small 2=Medium 3=Large");
        Console.Write("Please enter your selection: ");
        string s = Console.ReadLine();
        int n = int.Parse(s);
        int cost = 0;
        switch (n)
        {
            case 1:
                cost += 25;
                break;
            case 2:
                cost += 25;
                goto case 1;
            case 3:
                cost += 50;
                goto case 1;
            default:
                Console.WriteLine("Invalid selection.");
                break;
        }
        if (cost != 0)
        {
            Console.WriteLine("Please insert {0} cents.", cost);
        }
        Console.WriteLine("Thank you for your business.");
    }
}
```

输入

2

示例输出

```
Coffee sizes: 1=Small 2=Medium 3=Large
Please enter your selection: 2
Please insert 50 cents.
Thank you for your business.
```

下面的示例演示了使用 **goto** 跳出嵌套循环。

```
// statements_goto.cs
// Nested search loops
using System;
public class GotoTest1
{
```

```
static void Main()
{
    int x = 200, y = 4;
    int count = 0;
    string[,] array = new string[x, y];

    // Initialize the array:
    for (int i = 0; i < x; i++)

        for (int j = 0; j < y; j++)
            array[i, j] = (++count).ToString();

    // Read input:
    Console.Write("Enter the number to search for: ");

    // Input a string:
    string myNumber = Console.ReadLine();

    // Search:
    for (int i = 0; i < x; i++)
    {
        for (int j = 0; j < y; j++)
        {
            if (array[i, j].Equals(myNumber))
            {
                goto Found;
            }
        }
    }

    Console.WriteLine("The number {0} was not found.", myNumber);
    goto Finish;

    Found:
    Console.WriteLine("The number {0} is found.", myNumber);

    Finish:
    Console.WriteLine("End of search.");
}
}
```

输入

44

示例输出

```
Enter the number to search for: 44
The number 44 is found.
End of search.
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.10 Break、continue 和 goto 语句
- 8.9.3 goto 语句

请参见

参考

[C# 关键字](#)

[The goto Statement](#)

[跳转语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

return(C# 参考)

return 语句终止它出现在其中的方法的执行并将控制返回给调用方法。它还可以返回一个可选值。如果方法为 **void** 类型，则可以省略 **return** 语句。

示例

在下面的示例中，方法 `A()` 以 `double` 值的形式返回变量 `Area`。

```
// statements_return.cs
using System;
class ReturnTest
{
    static double CalculateArea(int r)
    {
        double area = r * r * Math.PI;
        return area;
    }

    static void Main()
    {
        int radius = 5;
        Console.WriteLine("The area is {0:0.00}", CalculateArea(radius));
    }
}
```

输出

The area is 78.54

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.12 Return 语句
- 8.9.4 return 语句

请参见

参考

[C# 关键字](#)

[The return Statement](#)

[跳转语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

异常处理语句 (C# 参考)

C# 为处理在程序执行期间可能出现的反常情况(称作异常)提供内置支持。这些异常由正常控制流之外的代码处理。

本节说明下列异常处理主题：

- [throw](#)
- [try-catch](#)
- [try-finally](#)
- [try-catch-finally](#)

请参见

参考

[C# 关键字](#)

[语句类型 \(C# 参考\)](#)

[异常和异常处理 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

throw(C# 参考)

throw 语句用于发出在程序执行期间出现反常情况(异常)的信号。

备注

引发的异常是一个对象，该对象的类是从 [System.Exception](#) 派生的，例如：

```
class MyException : System.Exception {}
// ...
throw new MyException();
```

通常 **throw** 语句与 try-catch 或 try-finally 语句一起使用。当引发异常时，程序查找处理此异常的 **catch** 语句。

也可以用 **throw** 语句重新引发已捕获的异常。有关更多信息和示例，请参见 [try-catch](#) 和 [引发异常](#)。

示例

此例演示如何使用 **throw** 语句引发异常。

```
// throw example
using System;
public class ThrowTest
{
    static void Main()
    {
        string s = null;

        if (s == null)
        {
            throw new ArgumentNullException();
        }

        Console.WriteLine("The string s is null"); // not executed
    }
}
```

输出

发生 [ArgumentNullException](#) 异常。

代码示例

请参见 [try-catch](#)、[try-finally](#) 和 [try-catch-finally](#) 示例。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.11 throw 语句
- 8.9.5 throw 语句

请参见

任务

[如何：显式引发异常](#)

参考

[The try, catch, and throw Statements](#)

[C# 关键字](#)

[异常处理语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

try-catch(C# 参考)

try-catch 语句由一个 **try** 块后跟一个或多个 **catch** 子句构成，这些子句指定不同的异常处理程序。

备注

try 块包含可能导致异常的保护代码。该块一直执行到引发异常或成功完成为止。例如，下列强制转换 **null** 对象的尝试引发 [NullReferenceException](#) 异常：

```
object o2 = null;
try
{
    int i2 = (int)o2;    // Error
}
```

catch 子句使用时可以不带任何参数，这种情况下它捕获任何类型的异常，并被称为一般 **catch** 子句。它还可以接受从 [System.Exception](#) 派生的对象参数，这种情况下它处理特定的异常。例如：

```
catch (InvalidCastException e)
{
}
```

在同一个 try-catch 语句中可以使用一个以上的特定 **catch** 子句。这种情况下 **catch** 子句的顺序很重要，因为会按顺序检查 **catch** 子句。将先捕获特定程度较高的异常，而不是特定程度较小的异常。

在 **catch** 块中可以使用 **throw** 语句再次引发已由 **catch** 语句捕获的异常。例如：

```
catch (InvalidCastException e)
{
    throw (e);    // Rethrowing exception e
}
```

如果要再次引发当前由无参数的 **catch** 子句处理的异常，则使用不带参数的 **throw** 语句。例如：

```
catch
{
    throw;
}
```

在 **try** 块内部时应该只初始化其中声明的变量；否则，完成该块的执行前可能发生异常。例如，在下面的代码示例中，变量 **x** 在 **try** 块内初始化。试图在 `Write(x)` 语句中的 **try** 块外部使用此变量时将产生编译器错误：[使用了未赋值的局部变量](#)。

```
static void Main()
{
    int x;
    try
    {
        // Don't initialize this variable here.
        x = 123;
    }
    catch
    {
    }
    // Error: Use of unassigned local variable 'x'.
    Console.Write(x);
}
```

有关 **catch** 块的更多信息，请参见 [try-catch-finally](#)。

示例

在此例中，**try** 块包含对可能导致异常的 `MyMethod()` 方法的调用。**catch** 子句包含仅在屏幕上显示消息的异常处理程序。当从 `MyMethod` 内部调用 **throw** 语句时，系统查找 **catch** 语句并显示 `Exception caught` 消息。

```
// try_catch_example.cs
using System;
class MainClass
{
    static void ProcessString(string s)
    {
        if (s == null)
        {
            throw new ArgumentNullException();
        }
    }

    static void Main()
    {
        try
        {
            string s = null;
            ProcessString(s);
        }
        catch (Exception e)
        {
            Console.WriteLine("{0} Exception caught.", e);
        }
    }
}
```

示例输出

```
System.ArgumentNullException: Value cannot be null.
at MainClass.Main() Exception caught.
```

此例使用了两个 `catch` 语句。最先出现的最特定的异常被捕获。

```
// try_catch_orderingCatchClauses.cs
using System;
class MainClass
{
    static void ProcessString(string s)
    {
        if (s == null)
        {
            throw new ArgumentNullException();
        }
    }

    static void Main()
    {
        try
        {
            string s = null;
            ProcessString(s);
        }
        // Most specific:
        catch (ArgumentNullException e)
        {
            Console.WriteLine("{0} First exception caught.", e);
        }
        // Least specific:
        catch (Exception e)
        {
            Console.WriteLine("{0} Second exception caught.", e);
        }
    }
}
```

示例输出

```
System.ArgumentNullException: Value cannot be null.  
at MainClass.Main() First exception caught.
```

注释

在前面的示例中，如果从具体程度最低的 catch 子句开始，您将收到以下错误信息：A previous catch clause already catches all exceptions of this or a super type ('System.Exception')。

但是，若要捕获特定程度最小的异常，请使用下面的语句替换 throw 语句：

```
throw new Exception();
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.13 Try-catch 语句
- 8.10 try 语句
- 16 异常

请参见

任务

[如何：显式引发异常](#)

参考

[C# 关键字](#)

[The try, catch, and throw Statements](#)

[异常处理语句\(C# 参考\)](#)

[throw\(C# 参考\)](#)

[try-finally\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

try-finally(C# 参考)

finally 块用于清除 [try](#) 块中分配的任何资源，以及运行任何即使在发生异常时也必须执行的代码。控制总是传递给 **finally** 块，与 [try](#) 块的退出方式无关。

备注

catch 用于处理语句块中出现的异常，而 **finally** 用于保证代码语句块的执行，与前面的 [try](#) 块的退出方式无关。

示例

在此例中，有一个导致异常的无效转换语句。当运行程序时，您收到一条运行时错误信息，但 **finally** 子句仍继续执行并显示输出。

```
// try-finally
using System;
public class MainClass
{
    static void Main()
    {
        int i = 123;
        string s = "Some string";
        object o = s;

        try
        {
            // Invalid conversion; o contains a string not an int
            i = (int)o;
        }
        finally
        {
            Console.WriteLine("i = {0}", i);
        }
    }
}
```

注释

上面的示例将导致引发 `System.InvalidCastException`。

尽管捕捉了异常，但仍会执行 **finally** 块中包含的输出语句，即：

`i = 123`

有关 **finally** 的更多信息，请参见 [try-catch-finally](#)。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.14 Try-finally 语句
- 8.11 try 语句
- 16 异常

请参见

任务

[如何：显式引发异常](#)

参考

[C# 关键字](#)

[The try, catch, and throw Statements](#)

[异常处理语句\(C# 参考\)](#)

[throw\(C# 参考\)](#)

[try-catch\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)
[C# 参考](#)

try-catch-finally(C# 参考)

catch 和 **finally** 一起使用的常见方式是: 在 **try** 块中获取并使用资源, 在 **catch** 块中处理异常情况, 并在 **finally** 块中释放资源。

有关重新引发异常的更多信息和示例, 请参见 [try-catch](#) 和[“引发异常”](#)。

示例

```
// try_catch_finally.cs
using System;
public class EHClass
{
    static void Main()
    {
        try
        {
            Console.WriteLine("Executing the try statement.");
            throw new NullReferenceException();
        }
        catch (NullReferenceException e)
        {
            Console.WriteLine("{0} Caught exception #1.", e);
        }
        catch
        {
            Console.WriteLine("Caught exception #2.");
        }
        finally
        {
            Console.WriteLine("Executing finally block.");
        }
    }
}
```

示例输出

```
Executing the try statement.
System.NullReferenceException: Object reference not set to an instance of an object.
  at EHClass.Main() Caught exception #1.
Executing finally block.
```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 5.3.3.15 Try-catch-finally 语句
- 8.10 try 语句
- 16 异常

请参见

任务

[如何: 显式引发异常](#)

参考

[C# 关键字](#)

[The try, catch, and throw Statements](#)

[异常处理语句\(C# 参考\)](#)

[throw\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

Checked 和 Unchecked(C# 参考)

C# 语句既可以在已检查的上下文中执行，也可以在未检查的上下文中执行。在已检查的上下文中，算法溢出引发异常。在未检查的上下文中，算法溢出被忽略并且结果被截断。

- [checked](#) 指定已检查的上下文。
- [unchecked](#) 指定未检查的上下文。

如果既未指定 **checked** 也未指定 **unchecked**，则默认上下文取决于外部因素(如编译器选项)。

下列操作受溢出检查的影响：

- 表达式在整型上使用下列预定义运算符：

`++` `-` `-(一元)` `+` `-` `*` `/`

- 整型间的显式数字转换。

[/checked](#) 编译器选项使您可以为 **checked** 或 **unchecked** 关键字范围内的所有非显式整型算术语句指定已检查或未检查的上下文。

请参见

参考

[C# 关键字](#)

[语句类型\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

checked(C# 参考)

checked 关键字用于对整型算术运算和转换显式启用溢出检查。

备注

默认情况下，如果表达式产生的值超出了目标类型的范围，则常数表达式将导致编译时错误，而非常数表达式在运行时计算并将引发异常。不过，如果通过编译器选项或环境配置在全局范围内取消了溢出检查，则可以使用 **checked** 关键字来启用此项功能。

请参见有关 **unchecked** 关键字用法的 [unchecked](#) 示例。

示例

此示例演示如何对非常数表达式使用 **checked**。在运行时会报告溢出。

```
// statements_checked.cs
using System;
class OverFlowTest
{
    static short x = 32767;    // Max short value
    static short y = 32767;

    // Using a checked expression
    static int CheckedMethod()
    {
        int z = 0;
        try
        {
            z = checked((short)(x + y));
        }
        catch (System.OverflowException e)
        {
            Console.WriteLine(e.ToString());
        }
        return z;
    }

    static void Main()
    {
        Console.WriteLine("Checked output value is: {0}",
                          CheckedMethod());
    }
}
```

示例输出

```
System.OverflowException: Arithmetic operation resulted in an overflow.
  at OverFlowTest.CheckedMethod()
Checked output value is: 0
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.2 Block 语句、checked 和 unchecked 语句
- 7.5.12 checked 和 unchecked 运算符
- 8.11 checked 和 unchecked 语句

请参见

参考

[C# 关键字](#)

[Checked 和 Unchecked\(C# 参考\)](#)

[unchecked\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

unchecked (C# 参考)

unchecked 关键字用于取消整型算术运算和转换的溢出检查。

备注

在未检查的上下文中，如果表达式产生目标类型范围之外的值，则结果被截断。例如：

```
unchecked
{
    int val = 2147483647 * 2;
}
```

因为上面的计算在 **unchecked** 块中执行，所以结果对于整数来说太大这一事实被忽略，并且 `val` 被赋予值 -2。默认情况下，启用溢出检测，这与使用 [checked](#) 具有相同的效果。

在上面的示例中，如果省略 **unchecked**，将产生编译错误，因为表达式使用常数，结果在编译时是已知的。**unchecked** 关键字还取消对非常数表达式的溢出检测，这是为了避免在运行时导致 [OverflowException](#)。

unchecked 关键字还可以用作运算符，如下所示：

```
public int uncheckedAdd(int a, int b)
{
    return unchecked(a + b);
}
```

示例

此示例通过在常数表达式中使用 **unchecked**，显示如何使用 **unchecked** 语句。

```
// statements_unchecked.cs
using System;

class TestClass
{
    const int x = 2147483647;    // Max int
    const int y = 2;

    static void Main()
    {
        int z;
        unchecked
        {
            z = x * y;
        }
        Console.WriteLine("Unchecked output value: {0}", z);
    }
}
```

输出

Unchecked output value: -2

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.3.3.2 Block 语句、`checked` 和 `unchecked` 语句
- 7.5.12 `checked` 和 `unchecked` 运算符
- 8.11 `checked` 和 `unchecked` 语句

请参见

[参考](#)

[C# 关键字](#)

[Checked 和 Unchecked\(C# 参考\)](#)

[checked\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

fixed 语句 (C# 参考)

fixed 语句禁止垃圾回收器重定位可移动的变量。**fixed** 语句只能出现在[不安全](#)的上下文中。**Fixed** 还可用于创建固定大小的缓冲区。

备注

fixed 语句设置指向托管变量的指针并在 *statement* 执行期间“钉住”该变量。如果没有 **fixed** 语句，则指向可移动托管变量的指针的作用很小，因为垃圾回收可能不可预知地重定位变量。C# 编译器只允许在 **fixed** 语句中分配指向托管变量的指针。

```
// assume class Point { public int x, y; }
// pt is a managed variable, subject to garbage collection.
Point pt = new Point();
// Using fixed allows the address of pt members to be
// taken, and "pins" pt so it isn't relocated.
fixed ( int* p = &pt.x )
{
    *p = 1;
}
```

可以用数组或字符串的地址初始化指针：

```
fixed (int* p = arr) ... // equivalent to p = &arr[0]
fixed (char* p = str) ... // equivalent to p = &str[0]
```

只要指针的类型相同，就可以初始化多个指针：

```
fixed (byte* ps = srcarray, pd = dstarray) {...}
```

要初始化不同类型的指针，只需嵌套 **fixed** 语句：

```
fixed (int* p1 = &p.x)
{
    fixed (double* p2 = &array[5])
    {
        // Do something with p1 and p2.
    }
}
```

执行完语句中的代码后，任何固定变量都被解除固定并受垃圾回收的制约。因此，不要指向 **fixed** 语句之外的那些变量。

注意

无法修改在 **fixed** 语句中初始化的指针。

在不安全模式中，可以在堆栈上分配内存。堆栈不受垃圾回收的制约，因此不需要被锁定。有关更多信息，请参见 [stackalloc](#)。

示例

```
// statements_fixed.cs
// compile with: /unsafe
using System;

class Point
{
    public int x, y;
}

class FixedTest
```

```
{  
    // Unsafe method: takes a pointer to an int.  
    unsafe static void SquarePtrParam (int* p)  
    {  
        *p *= *p;  
    }  
  
    unsafe static void Main()  
    {  
        Point pt = new Point();  
        pt.x = 5;  
        pt.y = 6;  
        // Pin pt in place:  
        fixed (int* p = &pt.x)  
        {  
            SquarePtrParam (p);  
        }  
        // pt now unpinned  
        Console.WriteLine ("{0} {1}", pt.x, pt.y);  
    }  
}
```

输出

25 6

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [18.3 固定变量和可移动变量](#)
- [18.6 fixed 语句](#)

请参见

参考

[C# 关键字](#)

[unsafe\(C# 参考\)](#)

[固定大小的缓冲区\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

lock 语句 (C# 参考)

lock 关键字将语句块标记为临界区，方法是获取给定对象的互斥锁，执行语句，然后释放该锁。此语句的形式如下：

```
Object thisLock = new Object();
lock (thisLock)
{
    // Critical code section
}
```

有关更多信息，请参见 [线程同步 \(C# 编程指南\)](#)。

备注

lock 确保当一个线程位于代码的临界区时，另一个线程不进入临界区。如果其他线程试图进入锁定的代码，则它将一直等待（即被阻止），直到该对象被释放。

[线程处理 \(C# 编程指南\)](#) 这节讨论了线程处理。

lock 调用块开始位置的 [Enter](#) 和块结束位置的 [Exit](#)。

通常，应避免锁定 **public** 类型，否则实例将超出代码的控制范围。常见的结构 `lock (this)`、`lock (typeof (MyType))` 和 `lock ("myLock")` 违反此准则：

- 如果实例可以被公共访问，将出现 `lock (this)` 问题。
- 如果 `MyType` 可以被公共访问，将出现 `lock (typeof (MyType))` 问题。
- 由于进程中使用同一字符串的任何其他代码将共享同一个锁，所以出现 `lock ("myLock")` 问题。

最佳做法是定义 **private** 对象来锁定，或 **private static** 对象变量来保护所有实例所共有的数据。

示例

下例显示的是在 C# 中使用线程的简单示例。

```
// statements_lock.cs
using System;
using System.Threading;

class ThreadTest
{
    public void RunMe()
    {
        Console.WriteLine("RunMe called");
    }

    static void Main()
    {
        ThreadTest b = new ThreadTest();
        Thread t = new Thread(b.RunMe);
        t.Start();
    }
}
```

输出

RunMe called

下例使用线程和 **lock**。只要 **lock** 语句存在，语句块就是临界区并且 `balance` 永远不会是负数。

```
// statements_lock2.cs
using System;
using System.Threading;
```

```

class Account
{
    private Object thisLock = new Object();
    int balance;

    Random r = new Random();

    public Account(int initial)
    {
        balance = initial;
    }

    int Withdraw(int amount)
    {

        // This condition will never be true unless the lock statement
        // is commented out:
        if (balance < 0)
        {
            throw new Exception("Negative Balance");
        }

        // Comment out the next line to see the effect of leaving out
        // the lock keyword:
        lock(thisLock)
        {
            if (balance >= amount)
            {
                Console.WriteLine("Balance before Withdrawal : " + balance);
                Console.WriteLine("Amount to Withdraw : -" + amount);
                balance = balance - amount;
                Console.WriteLine("Balance after Withdrawal : " + balance);
                return amount;
            }
            else
            {
                return 0; // transaction rejected
            }
        }
    }

    public void DoTransactions()
    {
        for (int i = 0; i < 100; i++)
        {
            Withdraw(r.Next(1, 100));
        }
    }
}

class Test
{
    static void Main()
    {
        Thread[] threads = new Thread[10];
        Account acc = new Account(1000);
        for (int i = 0; i < 10; i++)
        {
            Thread t = new Thread(new ThreadStart(acc.DoTransactions));
            threads[i] = t;
        }
        for (int i = 0; i < 10; i++)
        {
            threads[i].Start();
        }
    }
}

```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 5.3.3.18 Lock 语句
- 8.12 lock 语句

请参见

任务

[监视器同步技术示例](#)

[等待同步技术示例](#)

参考

[线程处理\(C# 编程指南\)](#)

[C# 关键字](#)

[语句类型\(C# 参考\)](#)

[MethodImplAttributes Enumeration](#)

[线程同步\(C# 编程指南\)](#)

[Mutex](#)

概念

[C# 编程指南](#)

[监视器](#)

[互锁操作](#)

[AutoResetEvent](#)

[其他资源](#)

[C# 参考](#)

方法参数(C# 参考)

如果在为方法声明参数时未使用 `ref` 或 `out`, 则该参数可以具有关联的值。可以在方法中更改该值, 但当控制传递回调用过程时, 不会保留更改的值。通过使用方法参数关键字, 可以更改这种行为。

本节描述声明方法参数时可以使用的关键字:

- [params](#)
- [ref](#)
- [out](#)

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

params(C# 参考)

params 关键字可以指定在参数数目可变处采用参数的[方法参数](#)。

在方法声明中的 **params** 关键字之后不允许任何其他参数，并且在方法声明中只允许一个 **params** 关键字。

示例

```
// cs_params.cs
using System;
public class MyClass
{
    public static void UseParams(params int[] list)
    {
        for (int i = 0 ; i < list.Length; i++)
        {
            Console.WriteLine(list[i]);
        }
        Console.WriteLine();
    }

    public static void UseParams2(params object[] list)
    {
        for (int i = 0 ; i < list.Length; i++)
        {
            Console.WriteLine(list[i]);
        }
        Console.WriteLine();
    }

    static void Main()
    {
        UseParams(1, 2, 3);
        UseParams2(1, 'a', "test");

        // An array of objects can also be passed, as long as
        // the array type matches the method being called.
        int[] myarray = new int[3] {10,11,12};
        UseParams(myarray);
    }
}
```

输出

```
1
2
3
```

```
1
a
test
```

```
10
11
12
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.5.1.4 参数数组

请参见

参考

[C# 关键字](#)

[方法参数\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

ref(C# 参考)

ref 关键字使参数按引用传递。其效果是，当控制权传递回调用方法时，在方法中对参数所做的任何更改都将反映在该变量中。若要使用 **ref** 参数，则方法定义和调用方法都必须显式使用 **ref** 关键字。例如：

```
class RefExample
{
    static void Method(ref int i)
    {
        i = 44;
    }
    static void Main()
    {
        int val = 0;
        Method(ref val);
        // val is now 44
    }
}
```

传递到 **ref** 参数的参数必须最先初始化。这与 **out** 不同，**out** 的参数在传递之前不需要显式初始化。（请参见 [out](#)。）

尽管 **ref** 和 **out** 在运行时的处理方式不同，但它们在编译时的处理方式是相同的。因此，如果一个方法采用 **ref** 参数，而另一个方法采用 **out** 参数，则无法重载这两个方法。例如，从编译的角度来看，以下代码中的两个方法是完全相同的，因此将不会编译以下代码：

```
class CS0663_Example
{
    // compiler error CS0663: "cannot define overloaded
    // methods that differ only on ref and out"
    public void SampleMethod(ref int i) { }
    public void SampleMethod(out int i) { }
}
```

但是，如果一个方法采用 **ref** 或 **out** 参数，而另一个方法不采用这两类参数，则可以进行重载，如下所示：

```
class RefOutOverloadExample
{
    public void SampleMethod(int i) { }
    public void SampleMethod(ref int i) { }
}
```

备注

属性不是变量，因此不能作为 **ref** 参数传递。

有关传递数组的信息，请参见 [使用 ref 和 out 传递数组](#)。

示例

按引用传递值类型（如上所示）是有用的，但是 **ref** 对于传递引用类型也是很有用的。这允许被调用的方法修改该引用所引用的对象，因为引用本身是按引用传递的。下面的示例显示出当引用类型作为 **ref** 参数传递时，可以更改对象本身。

```
class RefRefExample
{
    static void Method(ref string s)
    {
        s = "changed";
    }
    static void Main()
    {
        string str = "original";
        Method(ref str);
        // str is now "changed"
    }
}
```

```
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.1.5 引用参数
- 10.5.1.2 引用参数

请参见

参考

[传递参数\(C# 编程指南\)](#)

[方法参数\(C# 参考\)](#)

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

out(C# 参考)

out 关键字会导致参数通过引用来传递。这与 **ref** 关键字类似，不同之处在于 **ref** 要求变量必须在传递之前进行初始化。若要使用 **out** 参数，方法定义和调用方法都必须显式使用 **out** 关键字。例如：

```
class OutExample
{
    static void Method(out int i)
    {
        i = 44;
    }
    static void Main()
    {
        int value;
        Method(out value);
        // value is now 44
    }
}
```

尽管作为 **out** 参数传递的变量不需要在传递之前进行初始化，但需要调用方法以便在方法返回之前赋值。

ref 和 **out** 关键字在运行时的处理方式不同，但在编译时的处理方式相同。因此，如果一个方法采用 **ref** 参数，而另一个方法采用 **out** 参数，则无法重载这两个方法。例如，从编译的角度来看，以下代码中的两个方法是完全相同的，因此将不会编译以下代码：

```
class CS0663_Example
{
    // compiler error CS0663: "cannot define overloaded
    // methods that differ only on ref and out"
    public void SampleMethod(out int i) { }
    public void SampleMethod(ref int i) { }
}
```

但是，如果一个方法采用 **ref** 或 **out** 参数，而另一个方法不采用这两类参数，则可以进行重载，如下所示：

```
class RefOutOverloadExample
{
    public void SampleMethod(int i) { }
    public void SampleMethod(out int i) { }
}
```

备注

属性不是变量，因此不能作为 **out** 参数传递。

有关传递数组的信息，请参见[使用 ref 和 out 传递数组](#)。

示例

当希望方法返回多个值时，声明 **out** 方法很有用。使用 **out** 参数的方法仍然可以将变量用作返回类型（请参见 [return](#)），但它还可以将一个或多个对象作为 **out** 参数返回给调用方法。此示例使用 **out** 在一个方法调用中返回三个变量。请注意，第三个参数所赋的值为 Null。这样便允许方法有选择地返回值。

```
class OutReturnExample
{
    static void Method(out int i, out string s1, out string s2)
    {
        i = 44;
        s1 = "I've been returned";
        s2 = null;
    }
    static void Main()
    {
```

```
    int value;
    string str1, str2;
    Method(out value, out str1, out str2);
    // value is now 44
    // str1 is now "I've been returned"
    // str2 is (still) null;
}
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 5.1.6 输出参数
- 10.5.1.3 输出参数

请参见

参考

[C# 关键字](#)

[方法参数\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

命名空间关键字(C# 参考)

本节描述与 using 命名空间关联的关键字和运算符：

- [命名空间](#)
- [using](#)
- [. 运算符](#)
- [:: 运算符](#)
- [外部别名](#)

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

命名空间 (C# 参考)

namespace 关键字用于声明一个范围。此命名空间范围允许您组织代码并为您提供创建全局唯一类型的方法。

```
namespace SampleNamespace
{
    class SampleClass{}
    interface SampleInterface{}
    struct SampleStruct{}
    enum SampleEnum{a,b}
    delegate void SampleDelegate(int i);
    namespace SampleNamespace.Nested
    {
        class SampleClass2{}
    }
}
```

备注

在一个命名空间中，可以声明一个或多个下列类型：

- 另一个命名空间
- [类](#)
- [接口](#)
- [结构](#)
- [枚举](#)
- [委托](#)

无论您是否在 C# 源文件中显式声明了命名空间，编译器都会添加一个默认的命名空间。该未命名的命名空间（有时称为全局命名空间）存在于每一个文件中。全局命名空间中的任何标识符都可用于命名的命名空间中。

命名空间隐式具有公共访问权，并且这是不可修改的。有关可以分配给命名空间中的元素的访问修饰符的讨论，请参见[访问修饰符 \(C# 参考\)](#)。

在两个或更多的声明中定义一个命名空间是可以的。例如，下面的示例将两个类定义为 `MyCompany` 命名空间的一部分：

```
// cs_namespace_keyword.cs
// compile with: /target:library
namespace MyCompany.Proj1
{
    class MyClass
    {
    }
}

namespace MyCompany.Proj1
{
    class MyClass1
    {
    }
}
```

示例

下面的示例显示了如何在嵌套的命名空间中调用静态方法。

```
// cs_namespace_keyword_2.cs
using System;
namespace SomeNameSpace
{
```

```
public class MyClass
{
    static void Main()
    {
        Nested.NestedNameSpaceClass.SayHello();
    }
}

// a nested namespace
namespace Nested
{
    public class NestedNameSpaceClass
    {
        public static void SayHello()
        {
            Console.WriteLine("Hello");
        }
    }
}
```

输出

Hello

更多信息

有关使用命名空间的更多信息，请参见下列主题：

- [命名空间\(C# 编程指南\)](#)
- [使用命名空间\(C# 编程指南\)](#)
- [如何：使用命名空间别名限定符\(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 3.4.1 命名空间成员
- 3.8 命名空间和类型名称
- 9 命名空间

请参见

参考

[C# 关键字](#)

[命名空间关键字\(C# 参考\)](#)

[using\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

using(C# 参考)

using 关键字有两个主要用途：

- 作为指令，用于为命名空间创建别名或导入其他命名空间中定义的类型。请参见 [using 指令](#)。
- 作为语句，用于定义一个范围，在此范围的末尾将释放对象。请参见 [using 语句](#)。

[请参见](#)

[参考](#)

[C# 关键字](#)

[命名空间关键字\(C# 参考\)](#)

[extern\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

using 指令 (C# 参考)

using 指令有两个用途：

- 允许在命名空间中使用类型，这样，您就不必在该命名空间中限定某个类型的使用：

```
using System.Text;
```

- 为命名空间或类型创建别名。

```
using Project = PC.MyCompany.Project;
```

using 关键字还用来创建 **using** 语句，此类语句定义将在何时释放对象。有关更多信息，请参见 [using 语句](#)。

备注

using 指令的范围限制为包含它的文件。

创建 **using** 别名，以便更易于将标识符限定到命名空间或类型。

创建 **using** 指令，以便在命名空间中使用类型而不必指定命名空间。**using** 指令不为您提供对嵌套在指定命名空间中的任何命名空间的访问。

命名空间分为两类：用户定义的命名空间和系统定义的命名空间。用户定义的命名空间是在代码中定义的命名空间。若要查看系统定义的命名空间的列表，请参见 [.NET Framework 类库参考](#)。

有关引用其他程序集中的方法的示例，请参见 [创建和使用 C# DLL](#)。

示例 1

下面的示例显示了如何为命名空间定义和使用 **using** 别名：

```
namespace PC
{
    // Define an alias for the nested namespace.
    using Project = PC.MyCompany.Project;
    class A
    {
        void M()
        {
            // Use the alias
            Project.MyClass mc = new Project.MyClass();
        }
    }
    namespace MyCompany
    {
        namespace Project
        {
            public class MyClass{}
        }
    }
}
```

示例 2

下面的示例显示了如何为类定义 **using** 指令和 **using** 别名：

```
// cs_using_directive2.cs
// Using directive.
using System;
// Using alias for a class.
using AliasToMyClass = NameSpace1.MyClass;

namespace NameSpace1
```

```
{  
    public class MyClass  
    {  
        public override string ToString()  
        {  
            return "You are in NameSpace1.MyClass";  
        }  
    }  
}  
  
namespace NameSpace2  
{  
    class MyClass  
    {  
    }  
}  
  
namespace NameSpace3  
{  
    // Using directive:  
    using NameSpace1;  
    // Using directive:  
    using NameSpace2;  
  
    class MainClass  
    {  
        static void Main()  
        {  
            AliasToMyClass somevar = new AliasToMyClass();  
            Console.WriteLine(somevar);  
        }  
    }  
}
```

输出

You are in NameSpace1.MyClass

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 9.3 Using 指令

请参见

参考

[C# 关键字](#)

[命名空间关键字\(C# 参考\)](#)

[using 语句\(C# 参考\)](#)

概念

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

using 语句 (C# 参考)

定义一个范围，将在此范围之外释放一个或多个对象。

语法

```
using (Font font1 = new Font("Arial", 10.0f))
{
}
```

备注

C# 通过 .NET Framework 公共语言运行库 (CLR) 自动释放用于存储不再需要的对象的内存。内存的释放具有不确定性；一旦 CLR 决定执行垃圾回收，就会释放内存。但是，通常最好尽快释放诸如文件句柄和网络连接这样的有限资源。

using 语句允许程序员指定使用资源的对象应当何时释放资源。为 **using** 语句提供的对象必须实现 [IDisposable](#) 接口。此接口提供了 [Dispose](#) 方法，该方法将释放此对象的资源。

可以在到达 **using** 语句的末尾时，或者在该语句结束之前引发了异常并且控制权离开语句块时，退出 **using** 语句。

可以在 **using** 语句中声明对象（如上所示），或者在 **using** 语句之前声明对象，如下所示：

```
Font font2 = new Font("Arial", 10.0f);
using (font2)
{
    // use font2
}
```

可以有多个对象与 **using** 语句一起使用，但是必须在 **using** 语句内部声明这些对象，如下所示：

```
using (Font font3 = new Font("Arial", 10.0f),
       font4 = new Font("Arial", 10.0f))
{
    // Use font3 and font4.
}
```

示例

下面的示例显示用户定义类可以如何实现它自己的 **Dispose** 行为。注意类型必须从 [IDisposable](#) 继承。

```
using System;

class C : IDisposable
{
    public void UseLimitedResource()
    {
        Console.WriteLine("Using limited resource...");
    }

    void IDisposable.Dispose()
    {
        Console.WriteLine("Disposing limited resource.");
    }
}

class Program
{
    static void Main()
    {
        using (C c = new C())
        {
            c.UseLimitedResource();
        }
    }
}
```

```
        }
        Console.WriteLine("Now outside using statement.");
        Console.ReadLine();
    }
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [5.3.3.17 Using 语句](#)
- [8.13 using 语句](#)

请参见

参考

[C# 关键字](#)

[extern\(C# 参考\)](#)

[命名空间关键字\(C# 参考\)](#)

[using 指令\(C# 参考\)](#)

[实现 Finalize 和 Dispose 以清理非托管资源](#)

概念

[C# 编程指南](#)

[命名空间\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

外部别名 (C# 参考)

有时可能有必要引用具有相同完全限定类型的程序集的两个版本，例如当需要在同一应用程序中使用程序集的两个或更多的版本时。通过使用外部程序集别名，来自每个程序集的命名空间可以在由别名命名的根级别命名空间内包装，从而可在同一文件中使用。

注意

`extern` 关键字还用作方法修饰符，声明用非托管代码编写的方法。

若要引用两个具有相同完全限定类型的程序集，必须在命令行上指定别名，如下所示：

```
/r:GridV1=grid.dll  
/r:GridV2=grid20.dll
```

这将创建外部别名 GridV1 和 GridV2。若要从程序中使用这些别名，请使用 `extern` 关键字引用它们。例如：

```
extern alias GridV1;  
extern alias GridV2;
```

每一个外部别名声明都引入一个额外的根级别命名空间，它与全局命名空间平行，而不是在全局命名空间内。因此，来自每个程序集的类型就可以通过各自的、根源于适当的名空间别名的完全限定名来引用，而不会产生多义性。

在上面的示例中，`GridV1::Grid` 是来自 `grid.dll` 的网格控件，而 `GridV2::Grid` 是来自 `grid20.dll` 的网格控件。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 25.4 Extern 别名

请参见

参考

[C# 关键字](#)

[命名空间关键字 \(C# 参考\)](#)

[:: 运算符 \(C# 参考\)](#)

[/reference \(导入元数据\) \(C# 编译器选项\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

运算符关键字(C# 参考)

用于执行杂项操作，例如创建对象，检查对象的运行时类型，获取类型的大小，等等。本节介绍下列关键字：

- [as](#) 将对象转换为可兼容类型。
- [is](#) 检查对象的运行时类型。
- [new](#)
 - [new 运算符](#) 创建对象。
 - [new 修饰符](#) 隐藏继承成员。
 - [new 约束](#) 限定类型参数。
- [sizeof](#) 获取类型的大小。
- [typeof](#) 获取类型的 **System.Type** 对象。
- [true](#)
 - [true 运算符](#) 返回布尔值 true 表示真，否则返回 false。
 - [true](#) 表示布尔值 true。
- [false](#)
 - [false 运算符](#) 返回布尔值 true 表示假，否则返回 false。
 - [false](#) 表示布尔值 false。
- [stackalloc](#) 在堆栈上分配内存块。

在语句一节中介绍了下列可用作运算符和语句的关键字：

- [checked](#) 指定已检查的上下文。
- [unchecked](#) 指定未检查的上下文。

请参见

参考

[C# 关键字](#)

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

as(C# 参考)

用于在兼容的引用类型之间执行转换。例如：

```
string s = someObject as string;
if (s != null)
{
    // someObject is a string.
}
```

备注

as 运算符类似于强制转换操作；但是，如果转换不可行，**as** 会返回 **null** 而不是引发异常。更严格地说，这种形式的表达式

```
expression as type
```

等效于

```
expression is type ? (type)expression : (type)null
```

只是 `expression` 只被计算一次。

注意，**as** 运算符只执行引用转换和装箱转换。**as** 运算符无法执行其他转换，如用户定义的转换，这类转换应使用 `cast` 表达式来执行。

示例

```
// cs_keyword_as.cs
// The as operator.
using System;
class Class1
{
}

class Class2
{
}

class MainClass
{
    static void Main()
    {
        object[] objArray = new object[6];
        objArray[0] = new Class1();
        objArray[1] = new Class2();
        objArray[2] = "hello";
        objArray[3] = 123;
        objArray[4] = 123.4;
        objArray[5] = null;

        for (int i = 0; i < objArray.Length; ++i)
        {
            string s = objArray[i] as string;
            Console.Write("{0}:", i);
            if (s != null)
            {
                Console.WriteLine("'" + s + "'");
            }
            else
            {
                Console.WriteLine("not a string");
            }
        }
    }
}
```

```
    }  
}
```

输出

```
0:not a string  
1:not a string  
2:'hello'  
3:not a string  
4:not a string  
5:not a string
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 6 转换
- 7.9.10 as 运算符

请参见

参考

[C# 关键字](#)

[is\(C# 参考\)](#)

[?: 运算符\(C# 参考\)](#)

[运算符关键字\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

is(C# 参考)

检查对象是否与给定类型兼容。例如，可以确定对象是否与 **string** 类型兼容，如下所示：

```
if (obj is string)
{
}
```

备注

如果所提供的表达式非空，并且所提供的对象可以强制转换为所提供的类型而不会导致引发异常，则 **is** 表达式的计算结果将是 **true**。有关更多信息，请参见 [7.6.6 强制转换表达式](#)。

如果已知表达式将始终是 **true** 或始终是 **false**，则 **is** 关键字将导致编译时警告，但是，通常在运行时才计算类型兼容性。

不能重载 **is** 运算符。

请注意，**is** 运算符只考虑引用转换、装箱转换和取消装箱转换。不考虑其他转换，如用户定义的转换。

示例

```
// cs_keyword_is.cs
// The is operator.
using System;
class Class1
{
}
class Class2
{
}

class IsTest
{
    static void Test(object o)
    {
        Class1 a;
        Class2 b;

        if (o is Class1)
        {
            Console.WriteLine("o is Class1");
            a = (Class1)o;
            // Do something with "a."
        }
        else if (o is Class2)
        {
            Console.WriteLine("o is Class2");
            b = (Class2)o;
            // Do something with "b."
        }
        else
        {
            Console.WriteLine("o is neither Class1 nor Class2.");
        }
    }
    static void Main()
    {
        Class1 c1 = new Class1();
        Class2 c2 = new Class2();
        Test(c1);
        Test(c2);
        Test("a string");
    }
}
```

输出

- is Class1
- is Class2
- is neither Class1 nor Class2.

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 7.9.9 is 运算符

请参见

参考

[C# 关键字](#)

[typeof\(C# 参考\)](#)

[as\(C# 参考\)](#)

[运算符关键字\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

new(C# 参考)

在 C# 中, **new** 关键字可用作运算符、修饰符或约束。

[new 运算符](#)

用于创建对象和调用构造函数。

[new 修饰符](#)

用于向基类成员隐藏继承成员。

[new 约束](#)

用于在泛型声明中约束可能用作类型参数的参数的类型。

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

new 运算符 (C# 参考)

用于创建对象和调用构造函数。例如：

```
Class1 o = new Class1();
```

new 运算符还用于调用值类型的默认构造函数。例如：

```
int i = new int();
```

在上一个语句中，`i` 初始化为 0，它是 `int` 类型的默认值。该语句的效果等同于：

```
int i = 0;
```

有关默认值的完整列表，请参见[默认值表](#)。

请记住，为[结构声明](#)默认的构造函数是错误的，因为每一个值类型都隐式具有一个公共的默认构造函数。可以在结构类型上声明参数化构造函数以设置其初始值，但是，只有在需要默认值之外的值时才必须这样做。

值类型对象（例如结构）是在堆栈上创建的，而引用类型对象（例如类）是在堆上创建的。两种类型的对象都是自动销毁的，但是，基于值类型的对象是在超出范围时销毁，而基于引用类型的对象则是在对该对象的最后一个引用被移除之后在某个不确定的时间销毁。对于占用固定资源（例如大量内存、文件句柄或网络连接）的引用类型，有时需要使用确定性终止以确保对象被尽快销毁。有关更多信息，请参见[using 语句 \(C# 参考\)](#)。

不能重载 **new** 运算符。

如果 **new** 运算符分配内存失败，将引发异常 [OutOfMemoryException](#)。

示例

在下面的示例中，通过使用 **new** 运算符创建并初始化一个 `struct` 对象和一个类对象，然后为它们赋值。显示了默认值和所赋的值。

```
// cs_operator_new.cs
// The new operator.
using System;
struct SampleStruct
{
    public int x;
    public int y;

    public SampleStruct(int x, int y)
    {
        this.x = x;
        this.y = y;
    }
}

class SampleClass
{
    public string name;
    public int id;

    public SampleClass() {}

    public SampleClass(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
}

class MainClass
```

```
{  
    static void Main()  
    {  
        // Create objects using default constructors:  
        SampleStruct Location1 = new SampleStruct();  
        SampleClass Employee1 = new SampleClass();  
  
        // Display values:  
        Console.WriteLine("Default values:");  
        Console.WriteLine("    Struct members: {0}, {1}",  
                         Location1.x, Location1.y);  
        Console.WriteLine("    Class members: {0}, {1}",  
                         Employee1.name, Employee1.id);  
  
        // Create objects using parameterized constructors:  
        SampleStruct Location2 = new SampleStruct(10, 20);  
        SampleClass Employee2 = new SampleClass(1234, "John Martin Smith");  
  
        // Display values:  
        Console.WriteLine("Assigned values:");  
        Console.WriteLine("    Struct members: {0}, {1}",  
                         Location2.x, Location2.y);  
        Console.WriteLine("    Class members: {0}, {1}",  
                         Employee2.name, Employee2.id);  
    }  
}
```

输出

```
Default values:  
    Struct members: 0, 0  
    Class members: , 0  
Assigned values:  
    Struct members: 10, 20  
    Class members: John Martin Smith, 1234
```

注释

注意，在示例中字符串的默认值为 **null**，因此未显示它。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.5.10 new 运算符

请参见

参考

[C# 关键字](#)

[运算符关键字\(C# 参考\)](#)

[new\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

new 修饰符 (C# 参考)

在用作修饰符时, **new** 关键字可以显式隐藏从基类继承的成员。隐藏继承的成员意味着该成员的派生版本将替换基类版本。在不使用 **new** 修饰符的情况下隐藏成员是允许的, 但会生成警告。使用 **new** 显式隐藏成员会取消此警告, 并记录代之以派生版本这一事实。

若要隐藏继承的成员, 请使用相同名称在派生类中声明该成员, 并使用 **new** 修饰符修饰该成员。例如:

```
public class BaseC
{
    public int x;
    public void Invoke() {}
}
public class DerivedC : BaseC
{
    new public void Invoke() {}
}
```

在此示例中, `DerivedC.Invoke` 隐藏了 `BaseC.Invoke`。字段 `x` 不受影响, 因为它没有被类似名称的字段隐藏。

通过继承隐藏名称采用下列形式之一:

- 引入类或结构中的常数、指定、属性或类型隐藏具有相同名称的所有基类成员。
- 引入类或结构中的方法隐藏基类中具有相同名称的属性、字段和类型。同时也隐藏具有相同签名的所有基类方法。
- 引入类或结构中的索引器将隐藏具有相同名称的所有基类索引器。

对同一成员同时使用 **new** 和 **override** 是错误的, 因为这两个修饰符在含义上相互排斥。使用 **new** 会用同样的名称创建一个新成员并使原始成员变为隐藏的, 而 **override** 则扩展继承成员的实现。

在不隐藏继承成员的声明中使用 **new** 修饰符将会生成警告。

示例

在该例中, 基类 `BaseC` 和派生类 `DerivedC` 使用相同的字段名 `x`, 从而隐藏了继承字段的值。该示例演示了 **new** 修饰符的用法。另外还演示了如何使用完全限定名访问基类的隐藏成员。

```
// cs_modifier_new.cs
// The new modifier.
using System;
public class BaseC
{
    public static int x = 55;
    public static int y = 22;
}

public class DerivedC : BaseC
{
    // Hide field 'x'
    new public static int x = 100;

    static void Main()
    {
        // Display the new value of x:
        Console.WriteLine(x);
        // Display the hidden value of x:
        Console.WriteLine(BaseC.x);
        // Display the unhidden member y:
        Console.WriteLine(y);
    }
}
```

输出

在此示例中，嵌套类隐藏了基类中具有相同名称的类。该示例不仅演示了如何使用 **new** 修饰符来消除警告消息，而且还演示了如何使用完全限定名来访问隐藏的类成员。

```
// cs_modifier_new_nested.cs
// Using the new modifier with nested types.
using System;
public class BaseC
{
    public class NestedC
    {
        public int x = 200;
        public int y;
    }
}

public class DerivedC : BaseC
{
    // Nested type hiding the base type members.
    new public class NestedC
    {
        public int x = 100;
        public int y;
        public int z;
    }

    static void Main()
    {
        // Creating an object from the overlapping class:
        NestedC c1 = new NestedC();

        // Creating an object from the hidden class:
        BaseC.NestedC c2 = new BaseC.NestedC();

        Console.WriteLine(c1.x);
        Console.WriteLine(c2.x);
    }
}
```

输出

100
200

注释

如果移除 **new** 修饰符，该程序仍可编译和运行，但您会收到以下警告：

```
The keyword new is required on 'MyDerivedC.x' because it hides inherited member ' MyBaseC.x '
.
```

如果嵌套类型隐藏了另一个类型，也可以使用 **new** 修饰符来修改该嵌套类型，如下面的示例所示。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.2.2 new 修饰符

请参见

参考

[C# 关键字](#)

[运算符关键字\(C# 参考\)](#)

[修饰符\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

new 约束(C# 参考)

new 约束指定泛型类声明中的任何类型参数都必须有公共的无参数构造函数。当泛型类创建类型的新实例时，将此约束应用于类型参数，如下面的示例所示：

```
class ItemFactory<T> where T : new()
{
    public T GetNewItem()
    {
        return new T();
    }
}
```

当与其他约束一起使用时，**new()** 约束必须最后指定：

```
using System;
public class ItemFactory<T>
    where T : IComparable, new()
{
}
```

有关更多信息，请参见[类型参数的约束\(C# 编程指南\)](#)。

[C# 语言规范](#)

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 19.1.4 约束

请参见

参考

[C# 关键字](#)

[运算符关键字\(C# 参考\)](#)

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

[泛型\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

sizeof(C# 参考)

用于获取 [值类型](#) 的字节大小。例如，可以如下所示检索 **int** 类型的大小：

```
int intSize = sizeof(int);
```

备注

sizeof 运算符仅适用于值类型，而不适用于引用类型。

注意

从 C# 2.0 版开始，将 **sizeof** 应用于预定义类型不再要求使用[不安全模式](#)。

不能重载 **sizeof** 运算符。由 **sizeof** 运算符返回的值是 **int** 类型。下表显示表示某些预定义类型大小的常数值。

表达式	结果
<code>sizeof(sbyte)</code>	1
<code>sizeof(byte)</code>	1
<code>sizeof(short)</code>	2
<code>sizeof(ushort)</code>	2
<code>sizeof(int)</code>	4
<code>sizeof(uint)</code>	4
<code>sizeof(long)</code>	8
<code>sizeof(ulong)</code>	8
<code>sizeof(char)</code>	2 (Unicode)
<code>sizeof(float)</code>	4
<code>sizeof(double)</code>	8
<code>sizeof(bool)</code>	1

对于所有其他类型（包括 `struct`），**sizeof** 运算符只能在不安全代码块中使用。虽然可以使用 [SizeOf 方法](#)，但该方法返回的值和 **sizeof** 返回的值并不总是相同的。`Marshal.SizeOf` 在已封送处理类型后返回大小，而 **sizeof** 返回公共语言运行库分配的大小（包括任何空白）。

示例

```
// cs_operator_sizeof.cs
// compile with: /unsafe
using System;
class MainClass
{
    unsafe static void Main()
    {
        Console.WriteLine("The size of short is {0}.", sizeof(short));
        Console.WriteLine("The size of int is {0}.", sizeof(int));
```

```
        Console.WriteLine("The size of long is {0}.", sizeof(long));  
    }  
}
```

输出

The size of short is 2.
The size of int is 4.
The size of long is 8.

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 18.5.8 sizeof 运算符

请参见

参考

[C# 关键字](#)

[运算符关键字 \(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

typeof(C# 参考)

用于获取类型的 `System.Type` 对象。`typeof` 表达式采用以下形式：

```
System.Type type = typeof(int);
```

备注

若要获取表达式的运行时类型，可以使用 .NET Framework 方法 [GetType](#)，如下所示：

```
int i = 0;
System.Type type = i.GetType();
```

`typeof` 运算符也能用于公开的泛型类型。具有不止一个类型参数的类型的规范中必须有适当数量的逗号。不能重载 `typeof` 运算符。

示例

```
// cs_operator_typeof.cs
using System;
using System.Reflection;

public class SampleClass
{
    public int sampleMember;
    public void SampleMethod() {}

    static void Main()
    {
        Type t = typeof(SampleClass);
        // Alternatively, you could use
        // SampleClass obj = new SampleClass();
        // Type t = obj.GetType();

        Console.WriteLine("Methods:");
        MethodInfo[] methodInfo = t.GetMethods();

        foreach (MethodInfo mInfo in methodInfo)
            Console.WriteLine(mInfo.ToString());

        Console.WriteLine("Members:");
        MemberInfo[] memberInfo = t.GetMembers();

        foreach (MemberInfo mInfo in memberInfo)
            Console.WriteLine(mInfo.ToString());
    }
}
```

输出

```
Methods:
Void SampleMethod()
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
Members:
Void SampleMethod()
System.Type GetType()
System.String ToString()
Boolean Equals(System.Object)
Int32 GetHashCode()
Void .ctor()
Int32 sampleMember
```

此示例使用 **GetType** 方法确定用来包含数值计算的结果的类型。这取决于结果数字的存储要求。

```
// cs_operator_typeof2.cs
using System;
class GetTypeTest
{
    static void Main()
    {
        int radius = 3;
        Console.WriteLine("Area = {0}", radius * radius * Math.PI);
        Console.WriteLine("The type is {0}",
                          (radius * radius * Math.PI).GetType());
    }
}
```

输出

```
Area = 28.2743338823081
The type is System.Double
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.5.11 `typeof` 运算符

[请参见](#)

[参考](#)

[C# 关键字](#)

[is\(C# 参考\)](#)

[运算符关键字\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

true(C# 参考)

用作重载运算符或文本:

[true 运算符](#)

[true 标识符](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

true 运算符 (C# 参考)

当由用户定义的类型定义时，返回 `bool` 值 `true` 以表示真，否则返回 `false`。这对于表示 `true`、`false` 和 `null`(既非 `true` 也非 `false`) 的类型很有用，在数据库中使用了该运算符。

这些类型可用于控制 `if`、`do`、`while` 和 `for` 语句中以及条件表达式中的表达式。

如果类型定义了 `true` 运算符，它还必须定义 `false` 运算符。

类型不能直接重载条件逻辑运算符 (`&&` 和 `||`)，但通过重载规则逻辑运算符和 `true` 与 `false` 运算符可以达到同样的效果。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 10.9.1 一元运算符
- 7.11.2 用户定义的条件逻辑运算符
- 7.16 布尔表达式

请参见

参考

[C# 关键字](#)

[C# 运算符](#)

[false\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

true 字面常数 (C# 参考)

表示布尔值 true。

示例

```
// cs_keyword_true.cs
using System;
class TestClass
{
    static void Main()
    {
        bool a = true;
        Console.WriteLine( a ? "yes" : "no" );
    }
}
```

输出

yes

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 2.4.4.1 布尔文本

请参见

参考

[C# 关键字](#)

[false\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

false(C# 参考)

用作重载运算符或文本：

- [false 运算符](#)
- [false 标识符](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

false 运算符 (C# 参考)

返回布尔值 `true` 以指示假，否则返回 `false`。这对于数据库中使用的表示 `true`、`false` 和 `null`(既非 `true` 也非 `false`)的类型很有用。

这些类型可用于控制 `if`、`do`、`while` 和 `for` 语句中以及条件表达式中的表达式。

如果类型定义了 `false` 运算符，则它还必须定义 `true` 运算符。

类型不能直接重载条件逻辑运算符 `&&` 和 `||`，但通过重载正则逻辑运算符以及运算符 `true` 与 `false` 可以达到同样的效果。

C# 语言规范

有关更多信息，请参见 C# 语言规范 中的以下各章节：

- 10.9.1 一元运算符
- 7.11.2 用户定义的条件逻辑运算符
- 7.16 布尔表达式

请参见

参考

C# 关键字

C# 运算符

`true`(C# 参考)

概念

C# 编程指南

其他资源

C# 参考

false 字面常数 (C# 参考)

表示布尔值 false。

示例

```
// cs_keyword_false.cs
using System;
class TestClass
{
    static void Main()
    {
        bool a = false;
        Console.WriteLine( a ? "yes" : "no" );
    }
}
```

输出

no

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 2.4.4.1 布尔文本

请参见

参考

[C# 关键字](#)

[true\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

stackalloc(C# 参考)

在不安全的代码上下文中使用，可以在堆栈上分配内存块。

```
int* fib = stackalloc int[100];
```

备注

上面的示例在堆栈而不是堆上分配了一个内存块，它的大小足以包含 100 个 `int` 类型的元素；该块的地址存储在 `fib` 指针中。此内存不受垃圾回收的制约，因此不必将其钉住（通过 `fixed`）。内存块的生存期受定义它的方法的生存期的限制（没有在方法返回之前释放内存的途径）。

`stackalloc` 仅在局部变量的初始值设定项中有效。

由于涉及指针类型，`stackalloc` 要求不安全上下文。请参见[不安全代码和指针](#)。

`stackalloc` 类似于 C 运行时库中的 `_alloca`。

安全性

不安全代码是天生比非不安全替代代码安全性更低的代码。但是，通过使用 `stackalloc` 可以自动启用公共语言运行库 (CLR) 中的缓冲区溢出检测功能。如果检测到缓冲区溢出，进程将尽快终止，以最大限度地减小执行恶意代码的机会。

示例

```
// cs_keyword_stackalloc.cs
// compile with: /unsafe
using System;
class Test
{
    static unsafe void Main()
    {
        int* fib = stackalloc int[100];
        int* p = fib;
        *p++ = *p++ = 1;
        for (int i = 2; i < 100; ++i, ++p)
        {
            *p = p[-1] + p[-2];
        }
        for (int i = 0; i < 10; ++i)
        {
            Console.WriteLine(fib[i]);
        }
    }
}
```

输出

```
1
1
2
3
5
8
13
21
34
55
```

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 18.7 堆栈分配

请参见 参考

[C# 关键字](#)

[运算符关键字\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

[其他资源](#)

[C# 参考](#)

转换关键字(C# 参考)

本节描述在类型转换中使用的关键字：

- [explicit](#)
- [implicit](#)
- [operator](#)

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

explicit(C# 参考)

explicit 关键字用于声明必须使用强制转换来调用的用户定义的类型转换运算符。例如，在下面的示例中，此运算符将名为 Fahrenheit 的类转换为名为 Celsius 的类：

```
// Must be defined inside a class called Farenheit:
public static explicit operator Celsius(Farenheit f)
{
    return new Celsius((5.0f/9.0f)*(f.degrees-32));
}
```

可以如下所示调用此转换运算符：

```
Farenheit f = new Farenheit(100.0f);
Celsius c = (Celsius)f;
```

备注

转换运算符将源类型转换为目标类型。源类型提供转换运算符。与隐式转换不同，必须通过强制转换的方式来调用显式转换运算符。如果转换操作可能导致异常或丢失信息，则应将其标记为 **explicit**。这可以防止编译器无提示地调用可能产生无法预见后果的转换操作。

省略此强制转换将导致编译时错误[编译器错误 CS0266](#)。

有关更多信息，请参见[使用转换运算符\(C# 编程指南\)](#)。

示例

下面的示例提供 `Farenheit` 和 `Celsius` 类，它们中的每一个都为另一个提供显式转换运算符。

```
// cs_keyword_explicit_temp.cs
using System;
class Celsius
{
    public Celsius(float temp)
    {
        degrees = temp;
    }
    public static explicit operator Farenheit(Celsius c)
    {
        return new Farenheit((9.0f / 5.0f) * c.degrees + 32);
    }
    public float Degrees
    {
        get { return degrees; }
    }
    private float degrees;
}

class Farenheit
{
    public Farenheit(float temp)
    {
        degrees = temp;
    }
    public static explicit operator Celsius(Farenheit f)
    {
        return new Celsius((5.0f / 9.0f) * (f.degrees - 32));
    }
    public float Degrees
    {
        get { return degrees; }
    }
    private float degrees;
```

```

}

class MainClass
{
    static void Main()
    {
        Fahrenheit f = new Fahrenheit(100.0f);
        Console.WriteLine("{0} fahrenheit", f.Degrees);
        Celsius c = (Celsius)f;
        Console.WriteLine(" = {0} celsius", c.Degrees);
        Fahrenheit f2 = (Fahrenheit)c;
        Console.WriteLine(" = {0} fahrenheit", f2.Degrees);
    }
}

```

输出

```
100 fahrenheit = 37.77778 celsius = 100 fahrenheit
```

下面的示例定义一个结构 `Digit`, 该结构表示单个十进制数字。定义了一个运算符, 用于将 `byte` 转换为 `Digit`, 但因为并非所有字节都可以转换为 `Digit`, 所以该转换是显式的。

```

// cs_keyword_explicit_2.cs
using System;
struct Digit
{
    byte value;
    public Digit(byte value)
    {
        if (value > 9)
        {
            throw new ArgumentException();
        }
        this.value = value;
    }

    // Define explicit byte-to-Digit conversion operator:
    public static explicit operator Digit(byte b)
    {
        Digit d = new Digit(b);
        Console.WriteLine("conversion occurred");
        return d;
    }
}

class MainClass
{
    static void Main()
    {
        try
        {
            byte b = 3;
            Digit d = (Digit)b; // explicit conversion
        }
        catch (Exception e)
        {
            Console.WriteLine("{0} Exception caught.", e);
        }
    }
}

```

输出

```
conversion occurred
```

C# 语言规范

有关更多信息, 请参见 [C# 语言规范](#) 中的以下各章节:

- 6.2 显式转换

请参见

任务

[如何:在结构间实现用户定义的转换\(C# 编程指南\)](#)

参考

[C# 关键字](#)

[implicit\(C# 参考\)](#)

[operator\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

implicit(C# 参考)

implicit 关键字用于声明隐式的用户定义类型转换运算符。如果转换过程可以确保不会造成数据丢失，则可使用该关键字在用户定义类型和其他类型之间进行隐式转换。

```

class Digit
{
    public Digit(double d) { val = d; }
    public double val;
    // ...other members

    // User-defined conversion from Digit to double
    public static implicit operator double(Digit d)
    {
        return d.val;
    }
    // User-defined conversion from double to Digit
    public static implicit operator Digit(double d)
    {
        return new Digit(d);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Digit dig = new Digit(7);
        //This call invokes the implicit "double" operator
        double num = dig;
        //This call invokes the implicit "Digit" operator
        Digit dig2 = 12;
        Console.WriteLine("num = {0} dig2 = {1}", num, dig2.val);
        Console.ReadLine();
    }
}

```

备注

隐式转换可以通过消除不必要的类型转换来提高源代码的可读性。但是，因为隐式转换不需要程序员将一种类型显式强制转换为另一种类型，所以使用隐式转换时必须格外小心，以免出现意外结果。一般情况下，隐式转换运算符应当从不引发异常并且从不丢失信息，以便可以在程序员不知晓的情况下安全使用它们。如果转换运算符不能满足那些条件，则应将其标记为 **explicit**。有关更多信息，请参见[使用转换运算符](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#) 中的以下各章节：

- 6.1 隐式转换
- 10.9.3 转换运算符

请参见

任务

[如何：在结构间实现用户定义的转换\(C# 编程指南\)](#)

参考

[C# 关键字](#)

[explicit\(C# 参考\)](#)

[operator\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

operator(C# 参考)

使用 **operator** 关键字来重载内置运算符，或提供类或结构声明中的用户定义转换。

示例

下面是分数的一个极其简化的类。该类重载了 + 和 * 运算符，以执行分数加法和乘法；同时提供了将 Fraction 类型转换为 double 类型的转换运算符。

```
// cs_keyword_operator.cs
using System;
class Fraction
{
    int num, den;
    public Fraction(int num, int den)
    {
        this.num = num;
        this.den = den;
    }

    // overload operator +
    public static Fraction operator +(Fraction a, Fraction b)
    {
        return new Fraction(a.num * b.den + b.num * a.den,
            a.den * b.den);
    }

    // overload operator *
    public static Fraction operator *(Fraction a, Fraction b)
    {
        return new Fraction(a.num * b.num, a.den * b.den);
    }

    // user-defined conversion from Fraction to double
    public static implicit operator double(Fraction f)
    {
        return (double)f.num / f.den;
    }
}

class Test
{
    static void Main()
    {
        Fraction a = new Fraction(1, 2);
        Fraction b = new Fraction(3, 7);
        Fraction c = new Fraction(2, 3);
        Console.WriteLine((double)(a * b + c));
    }
}
```

输出

0.880952380952381

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.2.2 运算符重载
- 7.2.3 一元运算符重载决策
- 7.2.4 二元运算符重载决策

请参见

[任务](#)

[如何:在结构间实现用户定义的转换\(C# 编程指南\)](#)

[参考](#)

[C# 关键字](#)

[implicit\(C# 参考\)](#)

[explicit\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

访问关键字(C# 参考)

本节介绍下列访问关键字：

- [base](#)

访问基类的成员。

- [this](#)

引用类的当前实例。

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

base(C# 参考)

base 关键字用于从派生类中访问基类的成员：

- 调用基类上已被其他方法重写的方法。
- 指定创建派生类实例时应调用的基类构造函数。

基类访问只能在构造函数、实例方法或实例属性访问器中进行。

从静态方法中使用 **base** 关键字是错误的。

示例

在本例中，基类 Person 和派生类 Employee 都有一个名为 GetInfo 的方法。通过使用 **base** 关键字，可以从派生类中调用基类的 GetInfo 方法。

```
// keywords_base.cs
// Accessing base class members
using System;
public class Person
{
    protected string ssn = "444-55-6666";
    protected string name = "John L. Malgraine";

    public virtual void GetInfo()
    {
        Console.WriteLine("Name: {0}", name);
        Console.WriteLine("SSN: {0}", ssn);
    }
}
class Employee : Person
{
    public string id = "ABC567EFG";
    public override void GetInfo()
    {
        // Calling the base class GetInfo method:
        base.GetInfo();
        Console.WriteLine("Employee ID: {0}", id);
    }
}

class TestClass
{
    static void Main()
    {
        Employee E = new Employee();
        E.GetInfo();
    }
}
```

本示例显示如何指定在创建派生类实例时调用的基类构造函数。

```
// keywords_base2.cs
using System;
public class BaseClass
{
    int num;

    public BaseClass()
    {
        Console.WriteLine("in BaseClass()");
    }

    public BaseClass(int i)
```

```
    {
        num = i;
        Console.WriteLine("in BaseClass(int i)");
    }

    public int GetNum()
    {
        return num;
    }
}

public class DerivedClass : BaseClass
{
    // This constructor will call BaseClass.BaseClass()
    public DerivedClass() : base()
    {
    }

    // This constructor will call BaseClass.BaseClass(int i)
    public DerivedClass(int i) : base(i)
    {
    }

    static void Main()
    {
        DerivedClass md = new DerivedClass();
        DerivedClass md1 = new DerivedClass(1);
    }
}
```

输出

```
Name: John L. Malgraine
SSN: 444-55-6666
Employee ID: ABC567EFG
```

有关其他示例，请参见 [new、virtual 和 override](#)。

输出

```
in BaseClass()
in BaseClass(int i)
```

[C# 语言规范](#)

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1.6.3 基类](#)
- [7.5.8 基访问](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[this\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

this(C# 参考)

this 关键字引用类的当前实例。

以下是 **this** 的常用用途：

- 限定被相似的名称隐藏的成员，例如：

```
public Employee(string name, string alias)
{
    this.name = name;
    this.alias = alias;
}
```

- 将对象作为参数传递到其他方法，例如：

```
CalcTax(this);
```

- 声明索引器，例如：

```
public int this [int param]
{
    get { return array[param]; }
    set { array[param] = value; }
}
```

由于静态成员函数存在于类一级，并且不是对象的一部分，因此没有 **this** 指针。在静态方法中引用 **this** 是错误的。

示例

在本例中，**this** 用于限定 `Employee` 类成员 `name` 和 `alias`，它们都被相似的名称隐藏。**this** 还用于将对象传递到属于其他类的方法 `CalcTax`。

```
// keywords_this.cs
// this example
using System;
class Employee
{
    private string name;
    private string alias;
    private decimal salary = 3000.00m;

    // Constructor:
    public Employee(string name, string alias)
    {
        // Use this to qualify the fields, name and alias:
        this.name = name;
        this.alias = alias;
    }

    // Printing method:
    public void printEmployee()
    {
        Console.WriteLine("Name: {0}\nAlias: {1}", name, alias);
        // Passing the object to the CalcTax method by using this:
        Console.WriteLine("Taxes: {0:C}", Tax.CalcTax(this));
    }

    public decimal Salary
    {
        get { return salary; }
    }
}
```

```
        }
    }
class Tax
{
    public static decimal CalcTax(Employee E)
    {
        return 0.08m * E.Salary;
    }
}

class MainClass
{
    static void Main()
    {
        // Create objects:
        Employee E1 = new Employee("John M. Trainer", "jtrainer");

        // Display results:
        E1.printEmployee();
    }
}
```

输出

```
Name: John M. Trainer
Alias: jtrainer
Taxes: $240.00
```

有关其他示例，请参见 [class](#) 和 [struct](#)。

[C# 语言规范](#)

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [7.5.7 this 访问](#)
- [10.2.6.4 this 访问](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[base\(C# 参考\)](#)

[方法\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

文字关键字(C# 参考)

C# 有下列文字关键字：

- [null](#)
- [true](#)
- [false](#)
- [default](#)

请参见

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

null(C# 参考)

null 关键字是表示不引用任何对象的空引用的文字值。**null** 是引用类型变量的默认值。

C# 2.0 引入了可为空值的类型，这是可以设置成未定义值的数据类型。请参见[可空类型\(C# 编程指南\)](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 2.4.4.6 null 文本

请参见

参考

[C# 关键字](#)

[文字关键字\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

[默认值表\(C# 参考\)](#)

true(C# 参考)

用作重载运算符或文本:

[true 运算符](#)

[true 标识符](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

false(C# 参考)

用作重载运算符或文本：

- [false 运算符](#)
- [false 标识符](#)

[请参见](#)

[参考](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

default(C# 参考)

default 关键字。

default 关键字可在 **switch** 语句或泛型代码中使用。

- [switch 语句](#): 指定默认标签。
- [泛型代码](#): 指定类型参数的默认值。这对于引用类型为空，对于值类型为零。

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

上下文关键字(C# 参考)

上下文关键字用于提供代码中的特定含义，但它不是 C# 中的保留字。本节介绍下面这些上下文关键字：

`get` 为属性或索引器定义访问器方法。

`partial` 在整个同一编译单元内定义分部类、结构和接口。

`set` 为属性或索引器定义访问器方法。

`where` 向泛型声明中添加约束。

`yield` 在迭代器块中使用，用于向枚举数对象返回值或发信号结束迭代。

`value` 用于设置访问器和添加或移除事件处理程序。

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

get(C# 参考)

在属性或索引器中定义“访问器”方法，以检索该属性或该索引器元素的值。有关更多信息，请参见[属性](#)和[索引器](#)。

下面是名为 Seconds 的属性中的 **get** 访问器示例：

```
class TimePeriod
{
    private double _seconds;
    public double Seconds
    {
        get { return _seconds; }
        set { _seconds = value; }
    }
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.2 属性
- 10.6.2 访问器

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

partial(C# 参考)

分部类型定义允许将类、结构或接口的定义拆分到多个文件中。

在 File1.cs 中：

```
namespace PC
{
    partial class A { }
}
```

在 File2.cs 中：

```
namespace PC
{
    partial class A { }
}
```

备注

当使用大项目或自动生成的代码(如由[Windows 窗体设计器](#)提供的代码)时，将一个类、结构或接口类型拆分到多个文件中的做法就很有用。有关更多信息，请参见[分部类](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 19.4 分部类型
- 23 个分部类型

请参见

参考

[修饰符\(C# 参考\)](#)

[泛型介绍\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

set(C# 参考)

定义属性或索引器中的“访问器”方法，用于设置属性或索引器元素的值。有关更多信息，请参见[属性和索引器](#)。

这是名为 Seconds 的属性的 **set** 访问器的示例：

```
class TimePeriod
{
    private double _seconds;
    public double Seconds
    {
        get { return _seconds; }
        set { _seconds = value; }
    }
}
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.2 属性
- 10.6.2 访问器

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

value(C# 参考)

隐式参数 **value** 用于设置访问器以及添加或移除事件处理程序。

有关 **value** 用法的更多信息, 请参见[事件](#)和[访问器可访问性](#)。

C# 语言规范

有关更多信息, 请参见[C# 语言规范](#)中的以下各章节:

- 1.6.6.2 属性
- 10.6.2 访问器

请参见

参考

[C# 关键字](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

where(C# 参考)

where 子句用于指定类型约束，这些约束可以作为泛型声明中定义的类型参数的变量。例如，可以声明一个泛型类 MyGenericClass，这样，类型参数 T 就可以实现 **IComparable<T>** 接口：

```
public class MyGenericClass<T> where T:IComparable { }
```

除了接口约束，**where** 子句还可以包括基类约束，以指出某个类型必须将指定的类作为基类（或者就是该类本身），才能用作该泛型类型的类型参数。这样的约束一经使用，就必须出现在该类型参数的所有其他约束之前。

```
// cs_where.cs
// compile with: /target:library
using System;

class MyClassy<T, U>
    where T : class
    where U : struct
{
}
```

where 子句还可以包括构造函数约束。可以使用 new 运算符创建类型参数的实例；但类型参数为此必须受构造函数约束 **new()** 的约束。**new()** 约束可以让编译器知道：提供的任何类型参数都必须具有可访问的无参数（或默认）构造函数。例如：

```
// cs_where_2.cs
// compile with: /target:library
using System;
public class MyGenericClass <T> where T: IComparable, new()
{
    // The following line is not possible without new() constraint:
    T item = new T();
}
```

new() 约束出现在 **where** 子句的最后。

对于多个类型参数，每个类型参数都使用一个 **where** 子句，例如：

```
// cs_where_3.cs
// compile with: /target:library
using System;
using System.Collections;

interface MyI
{
}

class Dictionary<TKey, TValue>
    where TKey: IComparable, IEnumerable
    where TValue: MyI
{
    public void Add(TKey key, TValue val)
    {
    }
}
```

还可以将约束附加到泛型方法的类型参数，例如：

```
public bool MyMethod<T>(T t) where T : IMyInterface { }
```

请注意，对于委托和方法两者来说，描述类型参数约束的语法是一样的：

```
delegate T MyDelegate<T>() where T : new()
```

有关泛型委托的信息，请参见[泛型委托](#)。

有关约束的语法和用法的详细信息，请参见[类型参数的约束](#)。

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 19.1.4 约束

请参见

参考

[泛型介绍\(C# 编程指南\)](#)

[new 约束\(C# 参考\)](#)

[类型参数的约束\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

yield(C# 参考)

在迭代器块中用于向枚举数对象提供值或发出迭代结束信号。它的形式为下列之一：

```
yield return <expression>;
yield break;
```

备注

计算表达式并以枚举数对象值的形式返回；*expression* 必须可以隐式转换为迭代器的 yield 类型。

yield 语句只能出现在 **iterator** 块中，该块可用作方法、运算符或访问器的体。这类方法、运算符或访问器的体受以下约束的控制：

- 不允许不安全块。
- 方法、运算符或访问器的参数不能是 **ref** 或 **out**。

yield 语句不能出现在匿名方法中。有关更多信息，请参见[匿名方法\(C# 编程指南\)](#)。

当和 *expression* 一起使用时，**yield return** 语句不能出现在 **catch** 块中或含有一个或多个 **catch** 子句的 **try** 块中。有关更多信息，请参见[异常处理语句\(C# 参考\)](#)。

示例

在下面的示例中，迭代器块(这里是方法 `Power(int number, int power)`)中使用了 **yield** 语句。当调用 **Power** 方法时，它返回一个包含数字幂的可枚举对象。注意 **Power** 方法的返回类型是 **IEnumerable**(一种迭代器接口类型)。

```
// yield-example.cs
using System;
using System.Collections;
public class List
{
    public static IEnumerable Power(int number, int exponent)
    {
        int counter = 0;
        int result = 1;
        while (counter++ < exponent)
        {
            result = result * number;
            yield return result;
        }
    }

    static void Main()
    {
        // Display powers of 2 up to the exponent 8:
        foreach (int i in Power(2, 8))
        {
            Console.Write("{0} ", i);
        }
    }
}
```

输出

2 4 8 16 32 64 128 256

C# 语言规范

有关更多信息，请参见[C# 语言规范](#)中的以下各章节：

- 19.3 迭代器
- 22 迭代器

[请参见](#)

[参考](#)

[foreach, in\(C# 参考\)](#)

[使用迭代器\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

C# 运算符

C# 提供大量运算符，这些运算符是指定在表达式中执行哪些操作的符号。通常允许对枚举进行整型运算，例如 `==`、`!=`、`<`、`>`、`<=`、`>=`、**binary +**、**binary -**、**^**、**&**、**|**、**~**、**++**、**--** 和 **sizeof()**。此外，很多运算符可被用户重载，由此在应用到用户定义的类型时更改这些运算符的含义。

下表列出了按优先级顺序分组的 C# 运算符。每个组中的运算符具有相同的优先级。

运算符类别	运算符
基本	<code>x.y</code> <code>f(x)</code> <code>a[x]</code> <code>x++</code> <code>x--</code> <code>new</code> <code>typeof</code> <code>checked</code> <code>unchecked</code> <code>-></code>
一元	<code>+</code> <code>-</code> <code>!</code> <code>~</code> <code>++x</code> <code>--x</code> <code>(T)x</code> <code>True</code> <code>False</code> <code>&</code> <code>sizeof</code>
乘法	<code>*</code> <code>/</code> <code>%</code>
加法	<code>+</code> <code>-</code>
变换	<code><<</code> <code>>></code>

关系和类型检测	< > <=br/>>=br/>is as
相等	== !=
逻辑“与”	&
逻辑 XOR	^
逻辑“或”	
条件 AND	&&
条件 OR	
条件运算	:?
赋值	= += -= *= /= %= &= = ^= <<= >>= ??

算术溢出

算术运算符 (+、-、*、/) 产生的结果可能会超出涉及的数值类型可能值的范围。详细信息应参考有关特定运算符的章节，而一般情况下：

- 整数算术溢出或者引发 [OverflowException](#)，或者丢弃结果的最高有效位。整数被零除总是引发 [DivideByZeroException](#)。
- 浮点算术溢出或被零除从不引发异常，因为浮点类型基于 IEEE 754，因此可以表示无穷和 NaN（不是数字）。
- 小数**算术溢出总是引发 [OverflowException](#)。小数被零除总是引发 [DivideByZeroException](#)。

当发生整数溢出时，产生的结果取决于执行上下文，该上下文可为 [checked](#) 或 [unchecked](#)。在 checked 上下文中引发 [OverflowException](#)。在未选中的上下文中，放弃结果的最高有效位并继续执行。因此，C# 使您有机会选择处理或忽略溢出。

除算术运算符以外，整型之间的强制转换也会导致溢出（例如，将 long 强制转换为 int）并受 checked 或 unchecked 执行的限制。然而，按位运算符和移位运算符永远不会导致溢出。

[请参见](#)

[任务](#)

[“运算符重载”示例](#)

[参考](#)

[可重载运算符 \(C# 编程指南\)](#)

[C# 关键字](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

[Visual C#](#)

[] 运算符 (C# 参考)

方括号 (`[]`) 用于数组、索引器和属性，也可用于指针。

备注

数组类型是一种后跟 `[]` 的类型：

```
int[] fib; // fib is of type int[], "array of int"
fib = new int[100]; // create a 100-element int array
```

若要访问数组的一个元素，则用方括号括起所需元素的索引：

```
fib[0] = fib[1] = 1;
for( int i=2; i<100; ++i ) fib[i] = fib[i-1] + fib[i-2];
```

如果数组索引超出范围，则会引发异常。

不能重载数组索引运算符；但类型可以定义采用一个或多个参数的索引器和属性。索引器参数括在方括号中（就像数组索引一样），但索引器参数可声明为任何类型（这与数组索引不同，数组索引必须为整数）。

例如，.NET Framework 定义 **Hashtable** 类型，该类型将键和任意类型的值关联在一起。

```
Collections.Hashtable h = new Collections.Hashtable();
h["a"] = 123; // note: using a string as the index
```

方括号还用于指定属性 (C# 编程指南)：

```
[attribute(AllowMultiple=true)]
public class Attr
{
}
```

可以使用方括号来指定指针索引：

```
unsafe fixed ( int* p = fib ) // p points to fib from earlier example
{
    p[0] = p[1] = 1;
    for( int i=2; i<100; ++i ) p[i] = p[i-1] + p[i-2];
}
```

不执行边界检查。

C# 语言规范

有关更多信息，请参见 C# 语言规范 中的以下各章节：

- 1.6.6.5 运算符
- 7.2 运算符

请参见

参考

[C# 运算符](#)

[索引器 \(C# 编程指南\)](#)

[unsafe \(C# 参考\)](#)

[fixed 语句 \(C# 参考\)](#)

概念

[C# 编程指南](#)

[数组 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

() 运算符 (C# 参考)

除了用于指定表达式中的运算顺序外，圆括号还用于指定强制转换或类型转换：

```
double x = 1234.7;
int a;
a = (int)x; // cast double to int
```

备注

强制转换显式调用从一种类型到另一种类型的转换运算符；如果未定义这样的转换运算符，强制转换将失败。若要定义转换运算符，请参见 [explicit](#) 和 [implicit](#)。

不能重载 () 运算符。

有关更多信息，请参见 [强制转换 \(C# 编程指南\)](#)。

强制转换表达式可能导致不明确的语法。例如，表达式 $(x)-y$ 既可以解释为强制转换表达式（从类型 $-y$ 到类型 x 的强制转换），也可以解释为结合了带括号的表达式的相加表达式（计算 $x - y$ 的值）。

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [1.6.6.5 运算符](#)
- [7.2 运算符](#)

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

. 运算符 (C# 参考)

点运算符 (.) 用于成员访问。点运算符指定类型或命名空间的成员。例如，点运算符用于访问 .NET Framework 类库中的特定方法：

```
// The class Console in namespace System:  
System.Console.WriteLine("hello");
```

备注

例如，请考虑以下类：

```
class Simple  
{  
    public int a;  
    public void b()  
    {  
    }  
}  
Simple s = new Simple();
```

变量 s 有两个成员 a 和 b；若要访问这两个成员，请使用点运算符：

```
s.a = 6; // assign to field a;  
s.b(); // invoke member function b;
```

点还用于构造限定名，即指定其所属的命名空间或接口的名称。

```
// The class Console in namespace System:  
System.Console.WriteLine("hello");
```

using 指令使某些名称限定可选：

```
using System;  
// ...  
System.Console.WriteLine("hello");  
Console.WriteLine("hello"); // same thing
```

但是当某一标识符不确定时，必须限定它：

```
using System;  
// A namespace containing another Console class:  
using OtherSystem;  
// ...  
// Must qualify Console:  
System.Console.WriteLine( "hello" );
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.5.4 成员访问

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

:: 运算符 (C# 参考)

命名空间别名限定符运算符。

命名空间别名限定符 (::) 用于查找标识符。它通常放置在两个标识符之间，例如：

```
global::System.Console.WriteLine("Hello World");
```

备注

命名空间别名限定符可以是 global。这将调用全局命名空间中的查找，而不是在别名命名空间中。

更多信息

有关使用 :: 运算符的示例，请参见下面的章节：

- [如何：使用命名空间别名限定符 \(C# 编程指南\)](#)

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- [20.9.5 简单名称](#)
- [25.3 命名空间别名限定符](#)

请参见

参考

[C# 运算符](#)

[命名空间关键字 \(C# 参考\)](#)

[. 运算符 \(C# 参考\)](#)

[外部别名 \(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

+ 运算符 (C# 参考)

+ 运算符既可作为一元运算符也可作为二元运算符。

备注

一元 + 运算符是为所有数值类型预定义的。对数值类型进行一元 + 运算的结果就是操作数的值。

为数值类型和字符串类型预定义了二元 + 运算符。对于数值类型，+ 计算两个操作数之和。当其中的一个操作数是字符串类型或两个操作数都是字符串类型时，+ 将操作数的字符串表示形式串联在一起。

委托类型也提供二元 + 运算符，该运算符执行委托串联。

用户定义的类型可重载一元 + 运算符和二元 + 运算符(请参见 [operator](#))。在枚举时通常允许整型运算。

示例

```
// cs_operator_plus.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(+5);           // unary plus
        Console.WriteLine(5 + 5);        // addition
        Console.WriteLine(5 + .5);       // addition
        Console.WriteLine("5" + "5");   // string concatenation
        Console.WriteLine(5.0 + "5");   // string concatenation
        // note automatic conversion from double to string
    }
}
```

输出

```
5
10
5.5
55
55
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.5 运算符
- 7.2 运算符

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

- 运算符 (C# 参考)

- 运算符既可作为一元运算符也可作为二元运算符。

备注

一元 - 运算符是为所有数值类型预定义的。数值类型的一元 - 运算的结果是操作数的反数。

二元 - 运算符是为所有数值类型和枚举类型预定义的，其功能是从第一个操作数中减去第二个操作数。

委托类型也提供二元 - 运算符，该运算符执行委托移除。

用户定义的类型可重载一元 - 运算符和二元 - 运算符。有关更多信息，请参见 [operator](#)。

示例

```
// cs_operator_minus.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        Console.WriteLine(-a);
        Console.WriteLine(a - 1);
        Console.WriteLine(a - .5);
    }
}
```

输出

```
-5
4
4.5
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 1.6.6.5 运算符
- 7.2 运算符

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

* 运算符 (C# 参考)

乘法运算符 (*)，用于计算操作数的积。另外还用作取消引用运算符，允许读取和写入指针。

备注

所有数值类型都具有预定义的乘法运算符。

* 运算符还用来声明指针类型和取消引用指针。该运算符只能在不安全的上下文中使用，通过 `unsafe` 关键字的使用来表示，并且需要 `/unsafe` 编译器选项。取消引用运算符也称为间接寻址运算符。

用户定义的类型可重载二元 * 运算符(请参见 [operator](#))。重载二元运算符时，也会隐式重载相应的赋值运算符(如果有)。

示例

```
// cs_operator_mult.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(5 * 2);
        Console.WriteLine(-.5 * .2);
        Console.WriteLine(-.5m * .2m); // decimal type
    }
}
```

输出

```
10
-0.1
-0.10
```

```
// cs_operator_ptr.cs
// compile with: /unsafe
public class MainClass
{
    unsafe static void Main()
    {
        int i = 5;
        int* j = &i;
        System.Console.WriteLine(*j);
    }
}
```

输出

```
5
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

[不安全代码和指针\(C# 编程指南\)](#)

其他资源

[C# 参考](#)

/ 运算符 (C# 参考)

除法运算符 (/) 用第二个操作数除第一个操作数。所有数值类型都具有预定义的除法运算符。

备注

用户定义的类型可重载 / 运算符 (请参见[运算符](#))。重载 / 运算符将隐式重载 /= 运算符。

示例

```
// cs_operator_division.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(-5/2);
        Console.WriteLine(-5.0/2);
    }
}
```

输出

```
-2
-2.5
```

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

% 运算符 (C# 参考)

模数运算符 (%) 计算第二个操作数除第一个操作数后的余数。所有数值类型都具有预定义的模数运算符。

备注

用户定义的类型可重载 % 运算符 (请参见[运算符](#))。重载二元运算符时，也会隐式重载相应的赋值运算符 (如果有)。

示例

```
// cs_operator_modulus.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(5 % 2);          // int
        Console.WriteLine(-5 % 2);         // int
        Console.WriteLine(5.0 % 2.2);       // double
        Console.WriteLine(5.0m % 2.2m);     // decimal
        Console.WriteLine(-5.2 % 2.0);      // double
    }
}
```

输出

```
1
-1
0.6
0.6
-1.2
```

注释

请注意与双精度类型相关的舍入错误。

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

& 运算符 (C# 参考)

& 运算符既可作为一元运算符也可作为二元运算符。

备注

一元 & 运算符返回操作数的地址(要求 `unsafe` 上下文)。

为整型和 `bool` 类型预定义了二进制 & 运算符。对于整型, & 计算操作数的逻辑按位“与”。对于 `bool` 操作数, & 计算操作数的逻辑“与”；也就是说，当且仅当两个操作数均为 `true` 时，结果才为 `true`。

& 运算符计算两个运算符，与第一个操作数的值无关。例如：

```
int i = 0;
if (false & ++i == 1)
{
    // i is incremented, but the conditional
    // expression evaluates to false, so
    // this block does not execute.
}
```

用户定义的类型可重载二元 & 运算符(请参见 [operator](#))。在枚举时通常允许整型运算。重载二元运算符时，也会隐式重载相应的赋值运算符(如果有)。

示例

```
// cs_operator_ampersand.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true & false); // logical and
        Console.WriteLine(true & true); // logical and
        Console.WriteLine("0x{0:x}", 0xf8 & 0x3f); // bitwise and
    }
}
```

输出

```
False
True
0x38
```

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

| 运算符 (C# 参考)

二元 | 运算符是为整型和 **bool** 类型预定义的。对于整型，| 计算操作数的按位“或”结果。对于 **bool** 操作数，| 计算操作数的逻辑“或”结果；也就是说，当且仅当两个操作数均为 **false** 时，结果才为 **false**。

备注

用户定义的类型可重载 | 运算符 (请参见[运算符](#))。

示例

```
// cs_operator_OR.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true | false); // logical or
        Console.WriteLine(false | false); // logical or
        Console.WriteLine("0x{0:x}", 0xf8 | 0x3f); // bitwise or
    }
}
```

输出

True
False
0xff

请参见
参考
[C# 运算符](#)
[概念](#)
[C# 编程指南](#)
[其他资源](#)
[C# 参考](#)

^ 运算符 (C# 参考)

二元 `^` 运算符是为整型和 `bool` 类型预定义的。对于整型，`^` 将计算操作数的按位“异或”。对于 `bool` 操作数，`^` 将计算操作数的逻辑“异或”；也就是说，当且仅当只有一个操作数为 `true` 时，结果才为 `true`。

备注

用户定义的类型可重载 `^` 运算符(请参见[运算符](#))。在枚举时通常允许整型运算。

示例

```
// cs_operator_bitwise_OR.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(true ^ false); // logical exclusive-or
        Console.WriteLine(false ^ false); // logical exclusive-or
        // Bitwise exclusive-or:
        Console.WriteLine("0x{0:x}", 0xf8 ^ 0x3f);
    }
}
```

输出

```
True
False
0xc7
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

! 运算符 (C# 参考)

逻辑非运算符 (!) 是对操作数求反的一元运算符。为 **bool** 定义了该运算符，当且仅当操作数为 **false** 时才返回 **true**。

备注

用户定义的类型可重载 ! 运算符 (请参见[运算符](#))。

示例

```
// cs_operator_negation.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(!true);
        Console.WriteLine(!false);
    }
}
```

输出

False
True

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

~ 运算符 (C# 参考)

~ 运算符对操作数执行按位求补运算，其效果相当于反转每一位。按位求补运算符是为 [int](#)、[uint](#)、[long](#) 和 [ulong](#) 类型预定义的。

备注

用户定义的类型可重载 ~ 运算符。有关更多信息，请参见 [operator](#)。在枚举时通常允许整型运算。

示例

```
// cs_operator_bitwise_compl.cs
using System;
class MainClass
{
    static void Main()
    {
        int[] values = { 0, 0x111, 0xfffff, 0x8888, 0x22000022 };
        foreach (int v in values)
        {
            Console.WriteLine("~0x{0:x8} = 0x{1:x8}", v, ~v);
        }
    }
}
```

输出

```
~0x00000000 = 0xffffffff
~0x00000011 = 0xffffffffee
~0x000fffff = 0xffff0000
~0x00008888 = 0xffff7777
~0x22000022 = 0xddffffdd
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

= 运算符 (C# 参考)

赋值运算符 (=) 将右操作数的值存储在左操作数表示的存储位置、属性或索引器中，并将值作为结果返回。操作数的类型必须相同（或右边的操作数必须可以隐式转换为左边操作数的类型）。

备注

不能重载赋值运算符。

示例

```
// cs_operator_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        int i;
        i = 5; // int to int assignment
        x = i; // implicit conversion from int to double
        i = (int)x; // needs cast
        Console.WriteLine("i is {0}, x is {1}", i, x);
        object obj = i;
        Console.WriteLine("boxed value = {0}, type is {1}",
            obj, obj.GetType());
        i = (int)obj;
        Console.WriteLine("unboxed: {0}", i);
    }
}
```

输出

```
i is 5, x is 5
boxed value = 5, type is System.Int32
unboxed: 5
```

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

< 运算符 (C# 参考)

所有数值和枚举类型都定义“小于”关系运算符 (<)，如果第一个操作数小于第二个操作数，该运算符返回 **true**，否则返回 **false**。

备注

用户定义的类型可重载 < 运算符(请参见[运算符](#))。如果重载 <，则还必须重载 >。重载二元运算符时，也会隐式重载相应的赋值运算符(如果有)。

示例

```
// cs_operator_less_than.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1 < 1.1);
        Console.WriteLine(1.1 < 1.1);
    }
}
```

输出

```
True
False
```

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

> 运算符 (C# 参考)

所有数值类型和枚举类型都定义“大于”关系运算符 `>`，如果第一个操作数大于第二个操作数，它将返回 `true`，否则返回 `false`。

备注

用户定义的类型可重载 `>` 运算符 (请参见[运算符](#))。如果重载 `>`，则还必须重载 `<`。重载二元运算符时，也会隐式重载相应的赋值运算符 (如果有)。

示例

```
// cs_operator_greater_than.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1.1 > 1);
        Console.WriteLine(1.1 > 1.1);
    }
}
```

输出

True
False

[请参见](#)
[参考](#)

[C# 运算符](#)
[explicit\(C# 参考\)](#)
[概念](#)
[C# 编程指南](#)
[其他资源](#)
[C# 参考](#)

? : 运算符 (C# 参考)

条件运算符 (?) 根据布尔型表达式的值返回两个值中的一个。条件运算符的格式如下

```
condition ? first_expression : second_expression;
```

备注

如果条件为 **true**, 则计算第一表达式并以它的计算结果为准; 如果为 **false**, 则计算第二表达式并以它的计算结果为准。只计算两个表达式中的一个。

使用条件运算符, 可以更简洁、雅观地表达那些否则可能要求 if-else 结构的计算。例如, 为在 **sin** 函数的计算中避免被零除, 可编写为

```
if(x != 0.0) s = Math.Sin(x)/x; else s = 1.0;
```

或使用条件运算符,

```
s = x != 0.0 ? Math.Sin(x)/x : 1.0;
```

条件运算符为右联运算符, 因此该形式的表达式

```
a ? b : c ? d : e
```

按如下规则计算:

```
a ? b : (c ? d : e)
```

而不是按照下面这样计算:

```
(a ? b : c) ? d : e
```

不能重载条件运算符。

示例

```
// cs_operator_conditional.cs
using System;
class MainClass
{
    static double sinc(double x)
    {
        return x != 0.0 ? Math.Sin(x)/x : 1.0;
    }

    static void Main()
    {
        Console.WriteLine(sinc(0.2));
        Console.WriteLine(sinc(0.1));
        Console.WriteLine(sinc(0.0));
    }
}
```

输出

```
0.993346653975306
0.998334166468282
1
```

请参见

[参考](#)

[C# 运算符](#)

[if-else\(C# 参考\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

++ 运算符 (C# 参考)

增量运算符 (++) 将操作数加 1。增量运算符可以出现在操作数之前或之后：

备注

第一种形式是前缀增量操作。该操作的结果是操作数加 1 之后的值。

第二种形式是后缀增量操作。该运算的结果是操作数增加之前的值。

数值类型和枚举类型具有预定义的增量运算符。用户定义的类型可重载 ++ 运算符。在枚举时通常允许整型运算。

示例

```
// cs_operator_increment.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        x = 1.5;
        Console.WriteLine(++x);
        x = 1.5;
        Console.WriteLine(x++);
        Console.WriteLine(x);
    }
}
```

输出

```
2.5
1.5
2.5
```

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

-- 运算符 (C# 参考)

减量运算符 (--) 将操作数减 1。减量运算符可以出现在操作数之前或之后:--variable 和 variable--。第一种形式是前缀减量操作。该运算的结果是操作数减小“之后”的值。第二种形式是后缀减量操作。该运算的结果是操作数减小“之前”的值。

备注

数值类型和枚举类型具有预定义的增量运算符。

用户定义的类型可重载 -- 运算符(请参见[运算符](#))。在枚举时通常允许整型运算。

示例

```
// cs_operator_decrement.cs
using System;
class MainClass
{
    static void Main()
    {
        double x;
        x = 1.5;
        Console.WriteLine(--x);
        x = 1.5;
        Console.WriteLine(x--);
        Console.WriteLine(x);
    }
}
```

输出

0.5
1.5
0.5

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

&& 运算符 (C# 参考)

条件“与”运算符 (**&&**) 执行其 **bool** 操作数的逻辑“与”运算，但仅在必要时才计算第二个操作数。

备注

操作

```
x && y
```

对应于操作

```
x & y
```

不同的是，如果 **x** 为 **false**，则不计算 **y**(因为不论 **y** 为何值，“与”操作的结果都为 **false**)。这被称作为“短路”计算。

不能重载条件“与”运算符，但常规逻辑运算符和运算符 **true** 与 **false** 的重载，在某些限制条件下也被视为条件逻辑运算符的重载。

示例

在下面的示例中，请观察使用 **&&** 的表达式只计算第一个操作数。

```
// cs_operator_logical_and.cs
using System;
class MainClass
{
    static bool Method1()
    {
        Console.WriteLine("Method1 called");
        return false;
    }

    static bool Method2()
    {
        Console.WriteLine("Method2 called");
        return true;
    }

    static void Main()
    {
        Console.WriteLine("regular AND:");
        Console.WriteLine("result is {0}", Method1() & Method2());
        Console.WriteLine("short-circuit AND:");
        Console.WriteLine("result is {0}", Method1() && Method2());
    }
}
```

输出

```
regular AND:
Method1 called
Method2 called
result is False
short-circuit AND:
Method1 called
result is False
```

C# 语言规范

有关更多信息，请参见 [C# 语言规范](#) 中的以下各章节：

- 7.11.2 用户定义的条件逻辑运算符

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

|| 运算符 (C# 参考)

条件“或”运算符 (||) 执行 **bool** 操作数的逻辑“或”运算，但仅在必要时才计算第二个操作数。

备注

操作

```
x || y
```

对应于操作

```
x | y
```

不同的是，如果 `x` 为 `true`，则不计算 `y`(因为不论 `y` 为何值，“或”操作的结果都为 `true`)。这被称作为“短路”计算。

不能重载条件“或”运算符，但规则逻辑运算符和运算符 `true` 与 `false` 的重载，在某些限制条件下也被视为条件逻辑运算符的重载。

示例

在下面的示例中，请观察使用 `||` 的表达式只计算第一个操作数。

```
// cs_operator_short_circuit_OR.cs
using System;
class MainClass
{
    static bool Method1()
    {
        Console.WriteLine("Method1 called");
        return true;
    }

    static bool Method2()
    {
        Console.WriteLine("Method2 called");
        return false;
    }

    static void Main()
    {
        Console.WriteLine("regular OR:");
        Console.WriteLine("result is {0}", Method1() | Method2());
        Console.WriteLine("short-circuit OR:");
        Console.WriteLine("result is {0}", Method1() || Method2());
    }
}
```

输出

```
regular OR:
Method1 called
Method2 called
result is True
short-circuit OR:
Method1 called
result is True
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

<< 运算符 (C# 参考)

左移运算符 (<<) 将第一个操作数向左移动第二个操作数指定的位数。第二个操作数的类型必须是 [int](#)。

备注

如果第一个操作数是 [int](#) 或 [uint](#)(32 位数)，则移位数由第二个操作数的低 5 位给出。

如果第一个操作数是 [long](#) 或 [ulong](#)(64 位数)，则移位数由第二个操作数的低 6 位给出。

第一个操作数的高序位被放弃，低序空位用 0 填充。移位操作从不导致溢出。

用户定义的类型可重载 << 运算符(请参见 [operator](#))；第一个操作数的类型必须为用户定义的类型，第二个操作数的类型必须为 [int](#)。重载二元运算符时，也会隐式重载相应的赋值运算符(如果有)。

示例

```
// cs_operator_left_shift.cs
using System;
class MainClass
{
    static void Main()
    {
        int i = 1;
        long lg = 1;
        Console.WriteLine("0x{0:x}", i << 1);
        Console.WriteLine("0x{0:x}", i << 33);
        Console.WriteLine("0x{0:x}", lg << 33);
    }
}
```

输出

```
0x2
0x2
0x2000000000
```

注释

请注意，`i<<1` 和 `i<<33` 给出相同的结果，因为 1 和 33 的低序 5 个位相同。

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

>> 运算符 (C# 参考)

右移运算符 (>>) 将第一个操作数向右移动第二个操作数所指定的位数。

备注

如果第一个操作数为 [int](#) 或 [uint](#)(32 位数)，则移位数由第二个操作数的低五位给出(第二个操作数 & 0x1f)。

如果第一个操作数为 [long](#) 或 [ulong](#)(64 位数)，则移位数由第二个操作数的低六位给出(第二个操作数 & 0x3f)。

如果第一个操作数为 [int](#) 或 [long](#)，则右移位是算术移位(高序空位设置为符号位)。如果第一个操作数为 [uint](#) 或 [ulong](#) 类型，则右移位是逻辑移位(高位填充 0)。

用户定义的类型可重载 >> 运算符；第一个操作数的类型必须为用户定义的类型，第二个操作数的类型必须为 [int](#)。有关更多信息，请参见 [operator](#)。重载二元运算符时，也会隐式重载相应的赋值运算符(如果有)。

示例

```
// cs_operator_right_shift.cs
using System;
class MainClass
{
    static void Main()
    {
        int i = -1000;
        Console.WriteLine(i >> 3);
    }
}
```

输出

-125

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

== 运算符 (C# 参考)

对于预定义的值类型，如果操作数的值相等，则相等运算符 (==) 返回 `true`，否则返回 `false`。对于 `string` 以外的引用类型，如果两个操作数引用同一个对象，则 == 返回 `true`。对于 `string` 类型，== 比较字符串的值。

备注

用户定义的值类型可重载 == 运算符(请参见 [operator](#))。用户定义的引用类型也可重载 == 运算符，尽管在默认情况下，无论对于预定义的引用类型还是用户定义的引用类型，== 的行为都与上面描述的相同。如果重载 ==，则还必须重载 !=。在枚举时通常允许整型运算。

示例

```
// cs_operator_equality.cs
using System;
class MainClass
{
    static void Main()
    {
        // Numeric equality: True
        Console.WriteLine((2 + 2) == 4);

        // Reference equality: different objects,
        // same boxed value: False.
        object s = 1;
        object t = 1;
        Console.WriteLine(s == t);

        // Define some strings:
        string a = "hello";
        string b = String.Copy(a);
        string c = "hello";

        // Compare string values of a constant and an instance: True
        Console.WriteLine(a == b);

        // Compare string references;
        // a is a constant but b is an instance: False.
        Console.WriteLine((object)a == (object)b);

        // Compare string references, both constants
        // have the same value, so string interning
        // points to same reference: True.
        Console.WriteLine((object)a == (object)c);
    }
}
```

输出

```
True
False
True
False
True
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

!= 运算符 (C# 参考)

如果操作数相等，则不等运算符 (!=) 返回 `false`，否则，返回 `true`。为所有类型（包括字符串和对象）预定义了不等运算符。用户定义的类型可重载 `!=` 运算符。

备注

对于预定义的值类型，如果操作数的值不同，则不等运算符 (!=) 返回 `true`，否则，返回 `false`。对于 `string` 以外的引用类型，如果两个操作数引用不同的对象，则 `!=` 返回 `true`。对于 `string` 类型，`!=` 比较字符串的值。

用户定义的值类型可重载 `!=` 运算符（请参见 [operator](#)）。用户定义的引用类型也可重载 `!=` 运算符，尽管在默认情况下，无论对于预定义的引用类型还是用户定义的引用类型，`!=` 的行为都与上面描述的相同。如果重载 `!=`，则还必须重载 `==`。在枚举时通常允许整型运算。

示例

```
// cs_operator_inequality.cs
using System;
class MainClass
{
    static void Main()
    {
        // Numeric inequality:
        Console.WriteLine((2 + 2) != 4);

        // Reference equality: two objects, same boxed value
        object s = 1;
        object t = 1;
        Console.WriteLine(s != t);

        // String equality: same string value, same string objects
        string a = "hello";
        string b = "hello";

        // compare string values
        Console.WriteLine(a != b);

        // compare string references
        Console.WriteLine((object)a != (object)b);
    }
}
```

输出

```
False
True
False
False
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

<= 运算符 (C# 参考)

所有数值和枚举类型都定义了“小于等于”关系运算符 (`<=`)，如果第一个操作数小于或等于第二个操作数，则该运算符将返回 `true`，否则返回 `false`。

备注

用户定义的类型可重载 `<=` 运算符。有关更多信息，请参见 [operator](#)。如果重载 `<=`，则还必须重载 `>=`。在枚举时通常允许整型运算。

示例

```
// cs_operator_less_than_or_equal.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1 <= 1.1);
        Console.WriteLine(1.1 <= 1.1);
    }
}
```

输出

True
True

请参见

参考

[C# 运算符](#)

[explicit\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

>= 运算符 (C# 参考)

所有数值类型和枚举类型都定义“大于等于”关系运算符 `>=`，如果第一个操作数大于或等于第二个操作数，该运算符将返回 `true`，否则返回 `false`。

备注

用户定义的类型可重载 `>=` 运算符。有关更多信息，请参见 [operator](#)。如果重载 `>=`，则还必须重载 `<=`。在枚举时通常允许整型运算。

示例

```
// cs_operator_greater_than_or_equal.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine(1.1 >= 1);
        Console.WriteLine(1.1 >= 1.1);
    }
}
```

输出

True
True

[请参见](#)

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

+= 运算符 (C# 参考)

加法赋值运算符。

备注

使用 `+=` 赋值运算符的表达式，例如

```
x += y
```

等效于

```
x = x + y
```

不同的是 `x` 只计算一次。[+ 运算符](#) 的含义取决于 `x` 和 `y` 的类型(例如，对于数值操作数，其含义为相加；对于字符串操作数，其含义为串联)。

不能直接重载 `+=` 运算符，但用户定义的类型可以重载 [+ 运算符](#)(请参见 [operator](#))。

`+=` 运算符还用于指定响应事件时要调用的方法；这类方法称为事件处理程序。因为事件处理程序封装在委托类型中，所以在此上下文中使用 `+=` 运算符称为“委托串联”。有关更多信息，请参见 [event\(C# 参考\)](#) 和 [委托\(C# 编程指南\)](#)。

示例

```
// cs_operator_addition_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        //addition
        int a = 5;
        a += 6;
        Console.WriteLine(a);

        //string concatenation
        string s = "Micro";
        s += "soft";
        Console.WriteLine(s);
    }
}
```

输出

```
11
Microsoft
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

-= 运算符 (C# 参考)

减法赋值运算符。

备注

使用 -= 赋值运算符的表达式，如

```
x -= y
```

等效于

```
x = x - y
```

不同的是 x 只计算一次。- 运算符 的含义取决于 x 和 y 的类型(例如，对于数值操作数，其含义为相减；对于委托操作数，其含义为移除)。

不能直接重载 -= 运算符，但用户定义的类型可重载 - 运算符(请参见 [operator](#))。

示例

```
// cs_operator_subtraction_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a -= 6;
        Console.WriteLine(a);
    }
}
```

输出

-1

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

*= 运算符 (C# 参考)

二元乘法赋值运算符。

备注

使用 *= 赋值运算符的表达式，如

```
x *= y
```

等效于

```
x = x * y
```

不同的是 x 只计算一次。为数值类型预定义了 * 运算符以执行乘法操作。

不能直接重载 *= 运算符，但用户定义的类型可重载 * 运算符 (请参见 [operator](#))。

示例

```
// cs_operator_multiplication_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a *= 6;
        Console.WriteLine(a);
    }
}
```

输出

30

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

/= 运算符 (C# 参考)

除法赋值运算符。

备注

使用 /= 赋值运算符的表达式，如

```
x /= y
```

等效于

```
x = x / y
```

不同的是 x 只计算一次。为数值类型预定义了 [/ 运算符](#) 以执行除法操作。

不能直接重载 /= 运算符，但用户定义的类型可重载 [/ 运算符](#) (请参见 [operator](#))。对于所有复合赋值运算符，隐式重载二元运算符会重载等效的复合赋值。

示例

```
// cs_operator_division_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a /= 6;
        Console.WriteLine(a);
        double b = 5;
        b /= 6;
        Console.WriteLine(b);
    }
}
```

输出

```
0
0.8333333333333333
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

%= 运算符 (C# 参考)

模块赋值运算符。

备注

使用 %= 赋值运算符的表达式，如

```
x %= y
```

等效于

```
x = x % y
```

不同的是 x 只计算一次。为数值类型预定义了 [% 运算符](#)，以计算相除后的余数。

不能直接重载 %= 运算符，但用户定义的类型可重载 % 运算符(请参见[operator\(C# 参考\)](#))。

示例

```
// cs_operator_modulus_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 5;
        a %= 3;
        Console.WriteLine(a);
    }
}
```

输出

2

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

&= 运算符 (C# 参考)

“与”赋值运算符。

备注

使用 **&=** 赋值运算符的表达式，如

```
x &= y
```

等效于

```
x = x & y
```

不同的是 **x** 只计算一次。[& 运算符](#) 对整数操作数执行按位逻辑“与”运算，对 **bool** 操作数执行逻辑“与”运算。

不能直接重载 **&=** 运算符，但用户定义的类型可重载二元 [& 运算符](#) (请参见 [operator](#))。

示例

```
// cs_operator_and_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a &= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b &= false;
        Console.WriteLine(b);
    }
}
```

输出

0x00000004
False

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

|= 运算符 (C# 参考)

"或"赋值运算符。

备注

使用 |= 赋值运算符的表达式，例如

```
x |= y
```

等效于

```
x = x | y
```

不同的是 x 只计算一次。| 运算符对整型操作数执行按位逻辑"或"运算，对布尔操作数执行逻辑"或"运算。

不能直接重载 |= 运算符，但用户定义的类型可以重载 | 运算符 (请参见 [operator](#))。

示例

```
// cs_operator_or_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a |= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b |= false;
        Console.WriteLine(b);
    }
}
```

输出

```
0x0000000e
True
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

^= 运算符 (C# 参考)

"异或"赋值运算符。

备注

下列形式的表达式

```
x ^= y
```

按如下规则计算：

```
x = x ^ y
```

不同的是 `x` 只计算一次。[^ 运算符](#) 对整数操作数执行按位“异或”运算，对 `bool` 操作数执行逻辑“异或”运算。

不能直接重载 `^=` 运算符，但用户定义的类型可重载 [! 运算符](#) (请参见 [operator](#))。

示例

```
// cs_operator_xor_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 0x0c;
        a ^= 0x06;
        Console.WriteLine("0x{0:x8}", a);
        bool b = true;
        b ^= false;
        Console.WriteLine(b);
    }
}
```

输出

```
0x0000000a
True
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

<<= 运算符 (C# 参考)

左移赋值运算符。

备注

下列形式的表达式

```
x <<= y
```

按如下规则计算：

```
x = x << y
```

不同的是 `x` 只计算一次。[<< 运算符](#) 将 `x` 向左移动 `y` 指定的位数。

不能直接重载 `<<=` 运算符，但用户定义的类型可重载 [<< 运算符](#) (请参见 [operator](#))。

示例

```
// cs_operator_left_shift_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 1000;
        a <<= 4;
        Console.WriteLine(a);
    }
}
```

输出

16000

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

>>= 运算符 (C# 参考)

右移赋值运算符。

备注

下列形式的表达式

```
x >>= y
```

按如下规则计算：

```
x = x >> y
```

不同的是 `x` 只计算一次。[>> 运算符](#)根据 `y` 指定的量对 `x` 进行右移位。

不能直接重载 `>>=` 运算符，但用户定义的类型可重载 [>> 运算符](#)(请参见 [operator](#))。

示例

```
// cs_operator_right_shift_assignment.cs
using System;
class MainClass
{
    static void Main()
    {
        int a = 1000;
        a >>= 4;
        Console.WriteLine(a);
    }
}
```

输出

62

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

-> 运算符 (C# 参考)

-> 运算符将指针取消引用与成员访问组合在一起。

备注

以下形式的表达式

```
x->y
```

(其中 `x` 为 `T*` 类型的指针, `y` 为 `T` 的成员) 等效于

```
(*x).y
```

-> 运算符只能在[非托管代码](#)中使用。

不能重载 -> 运算符。

示例

```
// cs_operator_dereferencing.cs
// compile with: /unsafe
using System;
struct Point
{
    public int x, y;
}

class MainClass
{
    unsafe static void Main()
    {
        Point pt = new Point();
        Point* pp = &pt;
        pp->x = 123;
        pp->y = 456;
        Console.WriteLine ( "{0} {1}", pt.x, pt.y );
    }
}
```

输出

123 456

请参见

[参考](#)

[C# 运算符](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

?? 运算符 (C# 参考)

如果 ?? 运算符的左操作数非空，该运算符将返回左操作数，否则返回右操作数。

备注

可空类型可以包含值，或者可以是未定义的。?? 运算符定义当可空类型分配给非可空类型时返回的默认值。如果在将可空类型分配给非可空类型时不使用 ?? 运算符，将生成编译时错误。如果使用强制转换，并且当前未定义可空类型，将发生 **InvalidOperationException** 异常。

有关更多信息，请参见[可空类型 \(C# 编程指南\)](#)。

示例

```
// nullable_type_operator.cs
using System;
class MainClass
{
    static int? GetNullableInt()
    {
        return null;
    }

    static string GetStringValue()
    {
        return null;
    }

    static void Main()
    {
        // ?? operator example.
        int? x = null;

        // y = x, unless x is null, in which case y = -1.
        int y = x ?? -1;

        // Assign i to return value of method, unless
        // return value is null, in which case assign
        // default value of int to i.
        int i = GetNullableInt() ?? default(int);

        string s = GetStringValue();
        // ?? also works with reference types.
        // Display contents of s, unless s is null,
        // in which case display "Unspecified".
        Console.WriteLine(s ?? "Unspecified");
    }
}
```

请参见

参考

[C# 运算符](#)

概念

[C# 编程指南](#)

[可空类型 \(C# 编程指南\)](#)

其他资源

[C# 参考](#)

C# 预处理器指令

本节讨论 C# 语言的预处理器指令：

```
#if  
#else  
#elif  
#endif  
# define  
#undef  
#warning  
#error  
#line  
#region  
#endregion  
#pragma  
#pragma warning  
#pragma checksum
```

虽然编译器没有单独的预处理器，但在处理该节中描述的指令时如同存在一个单独的预处理器；这些指令用于辅助条件编译。与 C 和 C++ 指令不同，不能使用这些指令创建宏。

预处理器指令必须是行上的唯一指令。

[请参见](#)

[任务](#)

[“条件方法”示例](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

#if(C# 参考)

#if 使您可以开始条件指令，测试一个或多个符号以查看它们是否计算为 **true**。如果它们的计算结果确实为 **true**，则编译器将计算位于 **#if** 与最近的 **#endif** 指令之间的所有代码。例如，

```
#define DEBUG
// ...
#if DEBUG
    Console.WriteLine("Debug version");
#endif
```

可以使用运算符 **==**(相等)、**!=**(不相等)、**&&**(与)及 **||**(或)来计算多个符号。还可以用括号将符号和运算符分组。

备注

使用 **#if** 以及 **#else**、**#elif**、**#endif**、**#define** 和 **#undef** 指令，可以包括或排除基于由一个或多个符号组成的条件的代码。这在编译调试版本的代码或编译特定配置时最为有用。

以 **#if** 指令开始的条件指令必须用 **#endif** 指令显式终止。

#define 使您可以定义一个符号，通过将该符号用作传递给 **#if** 指令的表达式，使该表达式计算为 **true**。

也可以用 **/define** 编译器选项来定义符号。可以用 **#undef** 来取消定义符号。

用 **/define** 或 **#define** 定义的符号与具有同一名称的变量不冲突。即，不应将变量名传递到预处理器指令，并且只能用预处理器指令计算符号。

用 **#define** 创建的符号的范围是在其中定义该符号的文件。

示例

```
// preprocessor_if.cs
#define DEBUG#define VC_V7
using System;
public class MyClass
{
    static void Main()
    {
#if (DEBUG && !VC_V7)
        Console.WriteLine("DEBUG is defined");
#elif (!DEBUG && VC_V7)
        Console.WriteLine("VC_V7 is defined");
#elif (DEBUG && VC_V7)
        Console.WriteLine("DEBUG and VC_V7 are defined");
#else
        Console.WriteLine("DEBUG and VC_V7 are not defined");
#endif
    }
}
```

输出

DEBUG and VC_V7 are defined

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#else(C# 参考)

#else 允许您创建复合条件指令，因此，如果前面的 [#if](#) 或（可选）[#elif](#) 指令中的任何表达式都不为 `true`，则编译器将计算 **#else** 与后面的 [#endif](#) 之间的所有代码。

备注

[#endif](#) 必须是 **#else** 之后的下一条预处理器指令。有关如何使用 **#else** 的示例，请参见 [#if](#)。

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#elif(C# 参考)

#elif 使您得以创建复合条件指令。如果前面的 **#if** 和前面的任何 **#elif**(可选)指令表达式的计算结果都不是 **true**, 则将计算 **#elif** 表达式。如果 **#elif** 表达式计算为 **true**, 编译器将计算位于 **#elif** 和下一个条件指令之间的所有代码。例如:

```
#define VC7
//...
#if debug
    Console.WriteLine("Debug build");
#elif VC7
    Console.WriteLine("Visual Studio 7");
#endif
```

可以使用运算符 **==**(相等)、**!=**(不相等)、**&&**(与)及 **||**(或)来计算多个符号。还可以用括号将符号和运算符分组。

备注

#elif 等效于使用:

```
#else
#endif
```

使用 **#elif** 更简单, 因为每个 **#if** 都需要一个 **#endif**, 而 **#elif** 即使在没有匹配的 **#endif** 时也可以使用。

有关如何使用 **#elif** 的示例, 请参见 [#if](#)。

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#endif(C# 参考)

#endif 指定以 [#if](#) 指令开头的条件指令的结尾。例如，

```
#define DEBUG
// ...
#if DEBUG
    Console.WriteLine("Debug version");
#endif
```

备注

以 [#if](#) 指令开始的条件指令必须用 [#endif](#) 指令显式终止。有关如何使用 [#endif](#) 的示例，请参见 [#if\(C# 参考\)](#)。

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#define(C# 参考)

使用 **#define** 可以定义一个符号，并通过将该符号用作表达式传递给 **#if** 指令，使该表达式的计算结果为 **true**。例如：

```
# define DEBUG
```

备注

符号可用于指定编译的条件。可以使用 **#if** 或 **#elif** 来测试符号。还可以使用 **conditional** 属性执行条件编译。

可以定义符号，但是无法对符号赋值。**#define** 指令必须在使用任何也不是指令的指令之前出现在文件中。

也可以用 **/define** 编译器选项来定义符号。可以用 **#undef** 来取消定义符号。

用 **/define** 或 **#define** 定义的符号与具有同一名称的变量不冲突。即，不应将变量名传递到预处理器指令，并且只能用预处理器指令计算符号。

用 **#define** 创建的符号的范围是在其中定义该符号的文件。

有关如何使用 **#define** 的示例，请参见 [#if](#)。

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#undef(C# 参考)

#undef 使您可以取消符号的定义，以便通过将该符号用作 **#if** 指令中的表达式，使表达式的计算结果为 **false**。

可以使用 **#define** 指令或 **/define** 编译器选项定义符号。在使用任何不是指令的语句之前，必须在文件中使用 **#undef** 指令。

示例

```
// preprocessor_undef.cs
// compile with: /d:DEBUG
#undef DEBUG
using System;
class MyClass
{
    static void Main()
    {
#define DEBUG
        Console.WriteLine("DEBUG is defined");
#else
        Console.WriteLine("DEBUG is not defined");
#endif
    }
}
```

输出

```
DEBUG is not defined
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#warning (C# 参考)

#warning 使您得以从代码的特定位置生成一级警告。例如：

```
#warning Deprecated code in this method.
```

备注

#warning 通常用在条件指令中。也可以用 [#error \(C# 参考\)](#) 生成用户定义的错误。

示例

```
// preprocessor_warning.cs
// CS1030 expected
#define DEBUG
class MainClass
{
    static void Main()
    {
#if DEBUG
#warning DEBUG is defined
#endif
    }
}
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#error (C# 参考)

#error 使您可以从代码中的特定位置生成错误。例如：

```
#error Deprecated code in this method.
```

备注

#error 通常用在条件指令中。

也可以用 [#warning \(C# 参考\)](#) 生成用户定义的警告。

示例

```
// preprocessor_error.cs
// CS1029 expected
#define DEBUG
class MainClass
{
    static void Main()
    {
#if DEBUG
#error DEBUG is defined
#endif
    }
}
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#line(C# 参考)

#line 使您可以修改编译器的行号以及(可选)错误和警告的文件名输出。下面的示例说明如何报告与行号关联的两个警告。**#line 200** 指令强迫行号为 200(尽管默认值为 #7)。另一行 (#9) 作为默认 **#line** 指令的结果跟在通常序列后。

```
class MainClass
{
    static void Main()
    {
#line 200
        int i;      // CS0168 on line 200
#line default
        char c;     // CS0168 on line 9
    }
}
```

备注

#line 指令可能由生成过程中的自动中间步骤使用。例如, 如果行从原始的源代码文件中移除, 但是您仍希望编译器基于文件中的原始行号生成输出, 则可以移除行, 然后用 **#line** 模拟原始行号。

#line hidden 指令对调试器隐藏若干连续的行, 这样当开发人员在逐句通过代码时, 将会跳过 **#line hidden** 和下一个 **#line** 指令(假定它不是另一个 **#line hidden** 指令)之间的所有行。此选项也可用来使 ASP.NET 能够区分用户定义的代码和计算机生成的代码。尽管 ASP.NET 是此功能的主要使用者, 但很可能将有更多的源生成器使用它。

#line hidden 指令不会影响错误报告中的文件名或行号。即, 如果在隐藏块中遇到错误, 编译器将报告当前文件名和错误的行号。

#line filename 指令指定您希望出现在编译器输出中的文件名。默认情况下, 使用源代码文件的实际名称。文件名必须括在双引号 ("") 中。

源代码文件可以具有 **#line** 指令的任何编号。

示例 1

下面的示例说明调试器如何忽略代码中的隐藏行。运行此示例时, 它将显示三行文本。但是, 当设置如示例所示的断点并按 F10 键逐句通过代码时, 您将看到调试器忽略了隐藏行。还请注意, 即使在隐藏行上设置断点, 调试器仍会忽略它。

```
// preprocessor_linehidden.cs
using System;
class MainClass
{
    static void Main()
    {
        Console.WriteLine("Normal line #1."); // Set break point here.
#line hidden
        Console.WriteLine("Hidden line.");
#line default
        Console.WriteLine("Normal line #2.");
    }
}
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#region(C# 参考)

#region 使您可以在使用 Visual Studio 代码编辑器的[大纲显示](#)功能时指定可展开或折叠的代码块。例如：

```
#region MyClass definition
public class MyClass
{
    static void Main()
    {
    }
}
#endregion
```

备注

#region 块必须以 [#endregion](#) 指令终止。

#region 块不能与 [#if](#) 块重叠。但是，可以将 **#region** 块嵌套在 [#if](#) 块内，或将 [#if](#) 块嵌套在 **#region** 块内。

[请参见](#)

[参考](#)

[C# 预处理器指令](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

#endregion(C# 参考)

#endregion 标记 [#region](#) 块的结尾。例如：

```
#region MyClass definition
class MyClass
{
    static void Main()
    {
    }
}
#endif
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#pragma(C# 参考)

#pragma 用于给编辑器提供特殊的指令，说明如何编译包含杂注的文件。

```
#pragma pragma-name pragma-arguments
```

参数

pragma-name

可识别杂注的名称。

pragma-arguments

杂注特定的参数。

请参见

参考

[C# 预处理器指令](#)

[#pragma warning\(C# 参考\)](#)

[#pragma checksum\(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

#pragma warning(C# 参考)

#pragma warning 可用于启用或禁用某些警告。

```
#pragma warning disable warning-list
#pragma warning restore warning-list
```

参数

warning-list

警告编号的逗号分隔列表。只输入数字，不包括前缀 "CS"。

当没有指定警告编号时，**disable** 禁用所有警告，而 **restore** 启用所有警告。

示例

```
// pragma_warning.cs
using System;

#pragma warning disable 414, 3021
[CLSCompliant(false)]
public class C
{
    int i = 1;
    static void Main()
    {
    }
}
#pragma warning restore 3021
[CLSCompliant(false)] // CS3021
public class D
{
    int i = 1;
    public static void F()
    {
    }
}
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

#pragma checksum(C# 参考)

可用于生成源文件的校验和，以帮助调试 ASP.NET 页。

```
#pragma checksum "filename" "{guid}" "checksum bytes"
```

参数

"*filename*"

要求监视更改或更新的文件的名称。

"*{guid}*"

文件的全局唯一标识符 (GUID)。

"*checksum_bytes*"

十六进制数的字符串，表示校验和的字节。必须是偶数位的十六进制数。奇数位的数字会导致编译时警告，从而使指令被忽略。

备注

Visual Studio 调试器使用校验和来确保找到的总是正确的源。编译器计算源文件的校验和，然后将输出发出到程序数据库 (PDB) 文件。最后，调试器使用 PDB 来比较它为源文件计算的校验和。

此解决方案不适用于 ASP.NET 项目，因为算出的是生成的源文件而不是 .aspx 文件的校验和。为解决此问题，**#pragma checksum** 为 ASP.NET 页提供了校验和支持。

在 Visual C# 中创建 ASP.NET 项目时，生成的源文件包含 .aspx 文件(从该文件生成源文件)的校验和。然后，编译器将此信息写入 PDB 文件。

如果编译器在该文件中没有遇到 **#pragma checksum** 指令，它将计算校验和，然后将算出的值写入 PDB 文件。

示例

```
class TestClass
{
    static int Main()
    {
        #pragma checksum "file.cs" "{3673e4ca-6098-4ec1-890f-8fce2a794a2}" "{012345678AB}"
        // New checksum
    }
}
```

请参见

参考

[C# 预处理器指令](#)

概念

[C# 编程指南](#)

其他资源

[C# 参考](#)

C# 编译器选项

编译器产生可执行 (.exe) 文件、动态链接库文件 (.dll) 或者代码模块 (.netmodule)。

每个编译器选项均以两种形式提供：**-option** 和 **/option**。本文档仅显示 **/option** 形式。

本节内容

命令行生成

有关从命令行生成 Visual C# 应用程序的信息。

如何: 从命令行生成

提供运行 vsvars32.bat 以后用命令行编译的步骤。

部署 C# 应用程序

描述用于部署 C# 应用程序的选项。

按类别列出的 C# 编译器选项

编译器选项的分类列表。

按字母顺序列出的 C# 编译器选项

按字母顺序排序的编译器选项列表。

编译器错误

对 C# 编译器生成的错误的引用。

相关章节

如何: 设置项目属性 (C#、J#)

设置控制项目的编译、生成和调试方法的属性。包含有关在 Visual C# 项目中自定义生成步骤的信息。

默认与自定义生成

有关生成类型和配置的信息。

准备和管理生成

在 Visual Studio 开发环境内生成的过程。

命令行生成

可以通过在命令行上键入 C# 编译器的可执行文件 (csc.exe) 的名称来调用 C# 编译器。如果您使用“Visual Studio 命令提示”(以“开始”菜单上“Visual Studio 工具”下的快捷方式提供)，则系统已为您设置好所有必要的环境变量。否则，您可能需要调整路径，以调用将从计算机上的任何子目录中调用的 csc.exe。如果不使用“Visual Studio 命令提示”，则需要运行 vsvars32.bat 以设置适当的环境变量来支持命令行编译。有关 vsvars32.bat 的更多信息，请参见[如何：从命令行编译](#)。

如果您使用的计算机只安装有 .NET Framework SDK，则可在使用“SDK 命令提示”(可从“Microsoft .NET Framework SDK”菜单选项中访问)时在命令行上使用 C# 编译器。

若要从开发环境生成，请参见[准备和管理生成](#)。

csc.exe 可执行文件通常位于系统目录下的 Microsoft.NET\Framework\<version> 文件夹中。根据每台计算机上的确切配置，此位置可能有所不同。如果计算机上安装了 .NET Framework 的多个版本，则计算机上将存在此可执行文件的多个版本。有关这些安装的更多信息，请参见[安装 .NET Framework 的多个版本](#)。

本主题涵盖以下内容：

[命令行语法规则](#)

[命令行示例](#)

[C# 编译器和 C++ 编译器输出之间的差异](#)

命令行语法规则

在解释操作系统命令行上给出的参数时，C# 编译器使用下列规则：

- 参数用空白分隔，空白可以是一个空格或制表符。
- ^ 字符 (^) 未被识别为转义符或者分隔符。该字符在被传递给程序中的 argv 数组前，完全由操作系统的命令行分析器进行处理。
- 无论其中有无空白，包含在双引号 ("string") 中的字符串均被解释为单个参数。带引号的字符串可以嵌入在参数内。
- 前面有反斜杠的双引号 (\") 被解释为原义双引号字符 ("")。
- 反斜杠按其原义解释，除非它们紧位于双引号之前。
- 如果偶数个反斜杠后跟双引号，则每对反斜杠放置在 argv 数组中，并且双引号被解释为字符串分隔符。
- 如果奇数个反斜杠后跟双引号，则每对反斜杠放置在 argv 数组中，双引号由其余的反斜杠“转义”，使原义双引号 (") 被放置在 argv 数组中。

命令行示例

- 编译 File.cs 以产生 File.exe：

```
csc File.cs
```

- 编译 File.cs 以产生 File.dll：

```
csc /target:library File.cs
```

- 编译 File.cs 并创建 My.exe：

```
csc /out:My.exe File.cs
```

- 通过使用优化和定义 DEBUG 符号，编译当前目录中所有的 C# 文件。输出为 File2.exe：

```
csc /define:DEBUG /optimize /out:File2.exe *.cs
```

- 编译当前目录中所有的 C# 文件，以产生 File2.dll 的调试版本。不显示任何徽标和警告：

```
csc /target:library /out:File2.dll /warn:0 /nologo /debug *.cs
```

- 将当前目录中所有的 C# 文件编译为 Something.xyz(一个 DLL)：

```
csc /target:library /out:Something.xyz *.cs
```

C# 编译器和 C++ 编译器输出之间的差异

作为调用 C# 编译器的结果，没有创建任何对象 (.obj) 文件；直接创建输出文件。因此，C# 编译器不需要链接器。

[请参见](#)

[参考](#)

[按字母顺序列出的 C# 编译器选项](#)

[按类别列出的 C# 编译器选项](#)

[其他资源](#)

[C# 编译器选项](#)

如何：设置环境变量

vsvars32.bat 文件设置适当的环境变量以启用命令行编译。有关 vsvars32.bat 的更多信息，请参见下面的知识库文章：

- Q248802 : Vcvars32.bat Generates Out of Environment Message

如果安装 Visual Studio 的当前版本的计算机上已经有 Visual Studio 的早期版本，则不要在同一命令窗口中运行来自不同版本的 vsvars32.bat 或 vcvars32.bat。

运行 VSVARS32.BAT

1. 在命令提示处，更改到安装的 Common7\Tools 子目录。
2. 键入 VSVARS32 以运行 VSVARS32.bat。

⚠ 警告

VSVARS32.bat 可能因计算机的不同而变化。不要用其他计算机上的 VSVARS32.bat 替换丢失或损坏的 VSVARS32.bat 文件。重新运行安装程序以替换丢失的文件。

请参见

概念

命令行生成

部署 C# 应用程序

当您完成生成您的 C# 应用程序之后，下一步是分发 C# 应用程序。C# 是一种 .NET 语言；因此，将任何 C# 可执行文件分发到其他计算机上要求在每台执行计算机上安装 .NET Framework（可能还包括特定于您的应用程序的其他依赖项）。您有各种可用于分发 .NET Framework 的选项。有关概述，请参见[重新发布 .NET Framework](#)。

将完成的应用程序移动到其他计算机上的过程通常称为“部署”。Microsoft 的开发环境提供了部署机制；有关更多信息，请参见[部署应用程序和组件](#)。

如果您主要是从命令行生成和分发，您可能需要考虑其他部署和再分发依赖项的方法。

[请参见](#)

[概念](#)

[命令行生成](#)

按类别列出的 C# 编译器选项

下列编译器选项按类别排序。有关按字母顺序排序的列表，请参见[按字母顺序列出的 C# 编译器选项](#)。

优化

选项	用途
/filealign	指定输出文件中节的大小。
/optimize	启用/禁用优化。

输出文件

选项	用途
/doc	指定要将处理的文档注释写入到其中的 XML 文件。
/out	指定输出文件。
/pdb	指定 .pdb 文件的文件名和位置。
/platform	指定输出平台。
/target	使用下列四个选项之一指定输出文件的格式: /target:exe 、 /target:library 、 /target:module 或 /target:winexe

.NET Framework 程序集

选项	用途
/addmodule	指定一个或多个模块作为此程序集的一部分。
/delaysign	指示编译器添加公钥，但将此程序集保留为未签名状态。
/keycontainer	指定加密密钥容器的名称。
/keyfile	指定包含加密密钥的文件名。
/lib	指定通过 /reference 引用的程序集的位置。
/nostdlib	指示编译器不导入标准库 (mscorlib.dll)。
/reference	从包含程序集的文件中导入元数据。

调试/错误检查

选项	用途
/bugreport	创建一个文件，该文件包含有助于报告 bug 的信息。
/checked	指定溢出数据类型边界的整数算法是否将在运行时导致异常。
/debug	指示编译器发出调试信息。
/errorreport	设置错误报告行为。
/fullpaths	指定编译器输出中的文件的绝对路径。
/nowarn	取消编译器生成指定警告的功能。

/warn	设置警告等级。
/warnaserror	将警告提升为错误。

预处理器

选项	用途
/define	定义预处理器符号。

资源

选项	用途
/linkresource	创建到托管资源的链接。
/resource	将一个 .NET Framework 资源嵌入到输出文件中。
/win32icon	指定插入到输出文件中的.ico 文件。
/win32res	指定插入到输出文件中的 Win32 资源。

杂项

选项	用途
@	指定响应文件。
/?	将编译器选项列出到 stdout。
/baseaddress	指定加载 DLL 的首选地址。
/codepage	指定要用于编译中所有源代码文件的代码页。
/help	将编译器选项列出到 stdout。
/langversion	指定要使用的语言版本。
/main	指定 Main 方法的位置。
/noconfig	指示编译器不使用 csc.rsp 进行编译。
/nologo	不显示编译器版权标志信息。
/recurse	在子目录中搜索要编译的源文件。
/unsafe	允许编译使用 unsafe 关键字的代码。
/utf8output	使用 UTF-8 编码显示编译器输出。

已过时的选项

/incremental	启用增量编译。
--------------	---------

请参见

任务

[如何: 设置环境变量](#)

参考

[按字母顺序列出的 C# 编译器选项](#)

其他资源

[C# 编译器选项](#)

按字母顺序列出的 C# 编译器选项

下列编译器选项按字母顺序排序。有关分类列表，请参见[按类别列出的 C# 编译器选项](#)。

选项	用途
@	读取响应文件以获得更多选项。
/?	将用法信息显示到 stdout。
/addmodule	将指定的模块链接到此程序集中
/baseaddress	指定要生成的库的基址。
/bugreport	创建“Bug 报告”文件。如果与 /errorreport:prompt 或 /errorreport:send 一起使用，则此文件将与任何崩溃信息一起发送。
/checked	使编译器生成溢出检查。
/codepage	指定打开源文件时使用的代码页。
/debug	发出调试信息。
/define	定义条件编译符号。
/delaysign	仅使用强名称密钥的公共部分对程序集进行延迟签名。
/doc	指定要生成的 XML 文档文件。
/errorreport	指定如何处理内部编译器错误：prompt、send 或 none。默认值为 none。
/filealign	指定对输出文件节使用的对齐方式。
/fullpaths	使编译器生成完全限定路径。
/help	将用法信息显示到 stdout。
/incremental	启用增量编译 [obsolete]。
/keycontainer	指定强名称密钥容器。
/keyfile	指定强名称密钥文件。
/langversion	指定语言版本模式：ISO-1 或 Default。
/lib	指定要在其中搜索引用的附加目录。
/linkresource	将指定的资源链接到此程序集。
/main	指定包含入口点的类型（忽略所有其他可能的入口点）。
/noconfig	指示编译器不自动包含 CSC.RSP 文件。

/nologo	取消显示编译器版权信息。
/nostdlib	指示编译器不引用标准库 (mscorlib.dll)。
/nowarn	禁用特定的警告消息
/optimize	启用/禁用优化。
/out	指定输出文件名(默认值:包含主类的文件或第一个文件的基名称)。
/pdb	指定 .pdb 文件的文件名和位置。
/platform	限定此代码可以在其上运行的平台:x86、Itanium、x64 或 anycpu。默认值为 anycpu。
/recurse	按照通配符规范, 包括当前目录和子目录中的所有文件。
/reference	从指定的程序集文件引用元数据。
/resource	嵌入指定的资源。
/target	使用下列四个选项之一指定输出文件的格式: /target:exe 、 /target:library 、 /target:module 或 /target:winexe
/unsafe	允许使用 不安全 代码。
/utf8output	以 UTF-8 编码格式输出编译器消息。
/warn	设置警告等级 (0-4)。
/warnaserror	将特定的警告报告为错误。
/win32icon	对输出使用此图标。
/win32res	指定 Win32 资源文件 (.res)。

请参见

任务

[如何: 设置环境变量](#)

参考

[按类别列出的 C# 编译器选项](#)

其他资源

[C# 编译器选项](#)

@(指定响应文件)(C# 编译器选项)

@ 选项使您可以指定包含编译器选项和要编译的源代码文件的文件。

```
@response_file
```

参数

response_file

一个列出编译器选项或要编译的源代码文件的文件。

备注

编译器在处理编译器选项和源代码文件时将它们视为是在命令行上指定的。

若要在一次编译中指定多个响应文件，请指定多个响应文件选项。例如：

```
@file1.rsp @file2.rsp
```

在响应文件中，多个编译器选项和源代码文件可以出现在同一行中。单个编译器选项的指定必须出现在同一行中（不能跨行）。响应文件可以带有以 # 符号开始的注释。

从响应文件内部指定编译器选项就如同在命令行发出这些命令。有关更多信息，请参见[从命令行生成](#)。

编译器在遇到命令选项时会进行处理。因此，命令行参数可以重写先前在响应文件中列出的选项。反之，响应文件中的选项也将重写先前在命令行或其他响应文件中列出的选项。

C# 提供 csc.rsp 文件，该文件与 csc.exe 文件位于同一目录中。有关 csc.rsp 的更多信息，请参见[/noconfig](#)。

不能在 Visual Studio 开发环境中设置此编译器选项，也不能以编程方式对其进行更改。

示例

以下几行来自一个示例响应文件：

```
# build the first output file
/target:exe /out:MyExe.exe source1.cs source2.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/addmodule(导入元数据)(C# 编译器选项)

此选项将一个使用 `target:module` 开关创建的模块添加到当前的编译中。

```
/addmodule:[file];file2
```

参数

file, file2

包含元数据的输出文件。该文件不能包含程序集清单。若要导入多个文件，请用逗号或分号分隔文件名。

备注

运行时，所有用 `/addmodule` 添加的模块必须与输出文件位于同一目录中。也就是说，编译时可以在任何目录中指定模块，但是在运行时模块必须在应用程序目录中。如果模块在运行时不在应用程序目录中，您将遇到 [TypeLoadException](#)。

file 不能包含程序集。例如，如果输出文件用 `/target:module` 创建，则其元数据可以用 `/addmodule` 导入。

如果输出文件用 `/target` 选项而不是 `/target:module` 创建，则其元数据无法用 `/addmodule` 导入，但是可以用 `/reference` 导入。

此编译器选项在 Visual Studio 中不可用；项目不能引用模块。另外，不能以编程方式更改此编译器选项。

示例

编译源文件 `input.cs` 并从 `metad1.netmodule` 和 `metad2.netmodule` 中添加元数据来产生 `out.exe`：

```
csc /addmodule:metad1.netmodule;metad2.netmodule /out:out.exe input.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/baseaddress(指定 DLL 的基址)(C# 编译器选项)

/baseaddress 选项使您可以指定加载 DLL 的首选基址。

```
/baseaddress:address
```

参数

address

DLL 的基址。可以将该地址指定为十进制、十六进制或八进制数。

备注

DLL 的默认基址由 .NET Framework 公共语言运行库设置。

请注意：该地址中低位的数将会被舍入。例如，如果指定 0x11110001，它将被舍入为 0x11110000。

若要完成 DLL 的签名过程，请使用 SN.EXE 的 -R 选项。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“DLL 基址”属性。

若要以编程方式设置此编译器选项，请参见 [BaseAddress](#)。

请参见

[其他资源](#)

[C# 编译器选项](#)

/bugreport(报告问题)

指定应将调试信息放在文件中，供以后分析使用。

```
/bugreport:file
```

参数

file

要包含 bug 报告的文件的名称。

备注

/bugreport 选项指定应将下面的信息放在 *file* 中：

- 编译中所有源代码文件的副本。
- 编译中使用的编译器选项列表。
- 有关编译器、运行库和操作系统的版本信息。
- 引用的程序集和模块(保存为十六进制数)，.NET Framework 和 SDK 附带的程序集除外。
- 编译器输出(如果有)。
- 将会提示给您的问题的说明。
- 关于考虑问题应如何解决(就此将向您提示)的说明。

如果此选项与 **/errorreport:prompt** 或 **/errorreport:send** 一起使用，则文件中的信息将发送至 Microsoft Corporation。

由于所有源代码文件的副本都将放置在 *file* 中，因此可能需要在尽可能短的程序中重现所怀疑的代码缺陷。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

注意，生成文件的内容会公开源代码，可能导致意外的信息泄露。

请参见

参考

[/errorreport\(设置错误报告行为\)\(C# 编译器选项\)](#)

其他资源

[C# 编译器选项](#)

/checked(检查整数算法)(C# 编译器选项)

/checked 选项指定，不在 `checked` 或 `unchecked` 关键字的范围内、并且产生的值超出数据类型范围的整数算法语句是否将导致运行时异常。

```
/checked[+ | <U>-</U>]
```

备注

checked 或 **unchecked** 关键字范围内的整数算法语句不受 **/checked** 选项的影响。

如果不在 **checked** 或 **unchecked** 关键字范围内的整数算法语句产生的值超出数据类型范围，并且编译中使用了 **/checked+** (**/checked**)，则该语句将在运行时导致异常。如果编译中使用的是 **/checked-**，则该语句在运行时不会导致异常。

此选项的默认值是 **/checked-**，所以只需完全省略此选项，就可以获得相同的效果。使用 **/checked-** 的一个情况是在生成大应用程序时：有时使用自动化工具生成这类应用程序，这些工具可能自动将 **/checked** 设置为 **+**，这时您可以通过指定 **/checked-** 重写全局默认值。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关更多信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“检查算法上溢/下溢”属性。

若要以编程方式访问此编译器选项，请参见[CheckForOverflowUnderflow](#)。

示例

编译 `t2.cs`，并指定不在 **checked** 或 **unchecked** 关键字范围内并且产生的值超出数据类型范围的整数算法语句将在运行时导致异常。

```
csc t2.cs /checked
```

请参见

其他资源

[C# 编译器选项](#)

/codepage(指定源代码文件的代码页)(C# 编译器选项)

此选项指定在编译期间，当所需的页不是系统当前的默认代码页时使用的代码页。

```
/codepage:id
```

参数

id

编译中用于所有源代码文件的代码页的 ID。

备注

如果您编译的一个或多个源代码文件没有被创建为使用计算机上的默认代码页，您可以使用 **/codepage** 选项指定应使用哪个代码页。**/codepage** 适用于编译中的所有源代码文件。

如果源代码文件是用计算机中的同一有效代码页创建的，或者是用 UNICODE 或 UTF-8 创建的，则不需要使用 **/codepage**。

有关如何查找系统所支持的代码页的信息，请参见 [GetCPIInfo](#)。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

[其他资源](#)

[C# 编译器选项](#)

/debug(发出调试信息)(C# 编译器选项)

/debug 选项使编译器生成调试信息并将其放置在一个或多个输出文件中。

```
/debug[+ | <U>-</U>]
/debug:{<U>full</U> | pdbonly}
```

参数

+ | -

指定 + 或只指定 **/debug** 将使编译器生成调试信息，并将这些信息放置在一个程序数据库(.pdb 文件)中。指定 -(在不指定 **/debug** 时有效) 将导致不创建任何调试信息。

full | pdbonly

指定编译器生成的调试信息类型。full 参数在没有指定 **/debug:pdbonly** 时有效，它允许将调试器附加到正在运行的程序。

指定 pdbonly 允许在调试器中启动程序时进行源代码调试，但仅在正在运行的程序附加到调试器时才显示汇编程序。

备注

使用该选项创建调试版本。如果未指定 **/debug**、**/debug+** 或 **/debug:full**，则无法调试程序的输出文件。

如果使用 **/debug:full**，请注意 **/debug:full** 可能会对 JIT 优化代码的速度和大小有一些影响，并且可能会稍微影响代码质量。我们建议在生成发布代码时使用 **/debug:pdbonly** 或不使用 PDB。

注意

/debug:pdbonly 和 **/debug:full** 的一个区别是：使用 **/debug:full** 时，编译器会发出 [DebuggableAttribute](#)，用于通知 JIT 编译器有可用的调试信息。因此，如果使用 **/debug:full**，而代码中包含设置为 false 的 **DebuggableAttribute**，则会发生错误。

有关如何配置应用程序的调试性能的信息，请参见[使映像更易于调试](#)。

若要更改 .pdb 文件的位置，请参见[/pdb\(指定调试符号文件\)\(C# 编译器选项\)](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关更多信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“调试信息”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[DebugSymbols](#)。

示例

将调试信息放置在输出文件 app.pdb 中：

```
csc /debug /out:app.pdb test.cs
```

请参见

其他资源

[C# 编译器选项](#)

/define(预处理器定义)(C# 编译器选项)

/define 选项将 *name* 定义为程序中的符号。

```
/define:name[;name2]
```

参数

name, *name2*

要定义的一个或多个符号的名称。

备注

/define 选项的效果与在源文件中使用 `#define` 预处理器指令一样。符号一直保持定义到源文件中的 `#undef` 指令移除定义，或者编译器执行到文件尾。

可以将用该选项创建的符号与 `#if`、`#else`、`#elif` 和 `#endif` 一起使用，以按条件编译源文件。

/d 是 **/define** 的缩写形式。

通过使用分号或逗号分隔符号名称，可以使用 **/define** 定义多个符号。例如：

```
/define:DEBUG;TUESDAY
```

C# 编译器自身没有定义可以在源代码中使用的符号或宏；所有符号定义必须是用户定义的。

注意

与 C++ 这样的语言相同，C# `#define` 不允许给符号赋值。例如，不能使用 `#define` 创建宏或定义常数。如果您需要定义常数，请使用 `enum` 变量。如果您希望创建 C++ 样式的宏，请考虑其他方式，例如泛型。由于宏有容易出错的坏名声，因此 C# 不允许使用宏，而是提供了其他更安全的选择。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关更多信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“条件编译符号”属性。

有关如何以编程方式设置此编译器选项的信息，请参见 [DefineConstants](#)。

示例

```
// preprocessor_define.cs
// compile with: /define:xx
// or uncomment the next line
// #define xx
using System;
public class Test
{
    public static void Main()
    {
        #if (xx)
            Console.WriteLine("xx defined");
        #else
            Console.WriteLine("xx not defined");
        #endif
    }
}
```


/delaysign(延迟为程序集签名)(C# 编译器选项)

此选项将使编译器在输出文件中保留空间，以便以后添加数字签名。

```
/delaysign[ + | - ]
```

参数

+ | -

如果需要完全签名的程序集，则使用 **/delaysign-**。如果只想将公钥放在程序集中，则使用 **/delaysign+**。默认值为 **/delaysign-**。

备注

如果不与 [/keyfile](#) 或 [/keycontainer](#) 一起使用，**/delaysign** 选项将无效。

如果要求完全签名的程序集，编译器将对包含清单(程序集元数据)的文件进行散列处理，并用私钥对该散列数据进行签名。产生的数字签名存储在包含清单的文件中。当果程序集的签名延迟时，编译器将不会计算和存储签名，但会在文件中保留空间以便以后添加签名。

例如，使用 **/delaysign+** 允许测试人员将程序集放入全局缓存中。测试完成后，可以通过使用[程序集链接器](#)实用工具将私钥放入程序集中对程序集进行完全签名。

有关更多信息，请参见[创建和使用具有强名称的程序集](#)和[延迟为程序集签名](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关更多信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“签名”属性页。
3. 修改“仅延迟签名”属性。

有关如何以编程方式设置此编译器选项的信息，请参见 [DelaySign](#)。

请参见

[其他资源](#)

[C# 编译器选项](#)

/doc(处理文档注释)(C# 编译器选项)

/doc 选项允许在 XML 文件中放置文档注释。

```
/doc:file
```

参数

file

XML 的输出文件，由编译的源代码文件中的注释填充。

备注

在源代码文件中，可处理以下内容之前的文档注释，并将其添加到 XML 文件中：

- 用户定义的类型，如 [class](#)、[delegate](#) 或 [interface](#)
- 成员，如[字段](#)、[事件](#)、[属性](#)或方法

包含 Main 的源代码文件首先输出到 XML。

若要将生成的 .xml 文件用于 [Intellisense](#) 功能，请使该 .xml 文件的文件名与要支持的程序集同名，然后确保该 .xml 文件放入与该程序集相同的目录中。这样，在 Visual Studio 项目中引用程序集时，也可找到该 .xml 文件。有关更多信息，请参见[提供代码注释](#)。

除非用 [/target:module](#) 进行编译，否则 *file* 将包含 <assembly></assembly> 标记，以指定包含编译输出文件的程序集清单的文件名。

注意

/doc 选项适用于所有输入文件；或者，如果此选项是在“项目设置”中设置的，则适用于项目中的所有文件。若要禁用与特定的文件或代码段的文档注释相关的警告，请使用 [#pragma warning](#)。

有关从代码中的注释生成文档的方法，请参见[建议的文档注释标记](#)。

有关示例，请参见[“XML 文档”示例](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“XML 文档文件”属性。

有关如何以编程方式设置此编译器选项的信息，请参见 [DocumentationFile](#)。

请参见

任务

[“XML 文档”示例](#)

其他资源

[C# 编译器选项](#)

/errorreport(设置错误报告行为)(C# 编译器选项)

使用此选项可以方便地向 Microsoft 报告 C# 内部编译器错误。

```
/errorreport:{ none | prompt | queue | send }
```

参数

none

不收集或向 Microsoft 发送有关内部编译器错误的报告。

prompt

提示您在收到内部编译器错误时发送报告。在开发环境中编译应用程序时，**prompt** 是默认值。

queue

将错误报告加入队列。当使用管理员权限登录时，将打开一个弹出窗口并允许您报告自上次登录以来的任何失败（每三天内提示您发送失败报告的次数不会超过一次）。在命令行上编译应用程序时，**queue** 是默认值。

send

自动将内部编译器错误报告发送到 Microsoft。若要启用此选项，必须首先同意 Microsoft 的数据收集策略。首次在计算机上指定 **/errorreport:send** 时，编译器消息将引导您访问包含 Microsoft 的数据收集策略的网站。

备注

当编译器无法处理源代码文件时，将导致内部编译器错误 (ICE)。当发生 ICE 时，编译器不生成输出文件或可用来修复代码的任何有用的诊断。

在以前的版本中，当收到 ICE 时，最好致电 Microsoft 技术支持部门以报告问题。使用 **/errorreport**，您可以直接向 Visual C# 团队提供 ICE 信息。错误报告有助于改进未来的编译器版本。

用户发送报告的能力取决于计算机和用户策略权限。

有关错误调试器的更多信息，请参见 [Description of the Dr. Watson for Windows \(Drwtsn32.exe\) Tool](#) (Dr. Watson for Windows (Drwtsn32.exe) 工具的说明)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关更多信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“内部编译器错误报告”属性。

有关如何以编程方式设置此编译器选项的信息，请参见 [ErrorReport](#)。

请参见

其他资源

[C# 编译器选项](#)

/filealign(指定节对齐)(C# 编译器选项)

/filealign 选项使您可以指定输出文件中的节大小。

```
/filealign:number
```

参数

number

指定输出文件中节大小的值。有效值为 512、1024、2048、4096 和 8192。这些值以字节为单位。

备注

每个节将在是 **/filealign** 值的倍数的边界上对齐。没有固定的默认值。如果未指定 **/filealign**，则公共语言运行库在编译时将选取一个默认值。

通过指定节的大小，可以影响输出文件的大小。修改节的大小可能对将在较小设备上运行的程序有用。

使用 [DUMPPIN](#) 查看有关输出文件中的节信息。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“文件对齐”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[FileAlignment](#)。

请参见

[其他资源](#)

[C# 编译器选项](#)

/fullpaths(在编译器输出中指定完全限定路径)(C# 编译器选项)

/fullpaths 选项可使编译器在列出编译错误和警告时指定文件的完整路径。

```
/fullpaths
```

备注

默认情况下，编译所产生的错误和警告指定包含错误的文件名。**/fullpaths** 选项使编译器指定文件的完整路径。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

其他资源

[C# 编译器选项](#)

/help, /? (编译器命令行帮助) (C# 编译器选项)

该选项将编译器选项的列表和每个选项的简短说明发送到 stdout。

```
/help  
/?
```

备注

如果编译中包含该选项，将不会创建输出文件，也不会进行编译。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

其他资源

[C# 编译器选项](#)

/keycontainer(指定强名称密钥容器)(C# 编译器选项)

指定加密密钥容器的名称。

```
/keycontainer:string
```

参数

术语	定义
string	强名称密钥容器的名称。

备注

使用 /keycontainer 选项时，编译器将创建可共享的组件，方法是通过从指定的容器将公钥插入程序集清单中并使用私钥对最终程序集进行签名。若要生成密钥文件，请在命令行上键入 sn -k file。使用 sn -i 命令可将密钥对安装到容器中。

如果使用 [/target:module](#) 进行编译，则将密钥文件的名称保存在模块中，并在使用 [/addmodule](#) 编译程序集时将其包含到该程序集中。

在任何 Microsoft 中间语言 (MSIL) 模块的源代码中，也可以将此选项指定为一个自定义属性 ([System.Reflection.AssemblyKeyNameAttribute](#))。

也可以通过 [/keyfile](#) 将加密信息传递给编译器。如果希望将公钥添加到程序集清单中，但将程序集的签名延迟到该程序集通过测试，则请使用 [/delaysign](#)。

有关更多信息，请参见[创建和使用具有强名称的程序集](#)和[延迟为程序集签名](#)。

在 Visual Studio 开发环境中设置此编译器选项

- 此编译器选项在 Visual Studio 开发环境中不可用。

您可以使用 [AssemblyKeyContainerName](#) 以编程方式访问此编译器选项。

请参见

[其他资源](#)

[C# 编译器选项](#)

/keyfile(指定强名称密钥文件)(C# 编译器选项)

指定包含加密密钥的文件名。

```
/keyfile:file
```

参数

术语	定义
<i>file</i>	包含强名称密钥的文件的名称。

备注

使用此选项时，编译器将从指定的文件将公钥插入程序集清单中，然后使用私钥对最终程序集进行签名。若要生成密钥文件，请在命令行上键入 `sn -k file`。

如果使用 `/target:module` 进行编译，则将密钥文件的名称保存在模块中，并在使用 `/addmodule` 编译程序集时将其包含到创建的程序集中。

也可以使用 `/keycontainer` 将加密信息传递给编译器。如果需要部分签名的程序集，则使用 `/delaysign`。

如果在相同的编译中指定了 `/keyfile` 和 `/keycontainer`(通过命令行选项或自定义属性)，编译器将首先尝试密钥容器。如果成功，则使用密钥容器中的信息对程序集进行签名。如果编译器没有找到密钥容器，则将尝试用 `/keyfile` 指定的文件。如果成功，则使用密钥文件中的信息对程序集进行签名，并且将把密钥信息安装到密钥容器中(类似 `sn -i`)，这样，下次编译时密钥容器将是有效的。

请注意，密钥文件可能只包含公钥。

有关更多信息，请参见[创建和使用具有强名称的程序集](#)和[延迟为程序集签名](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“签名”属性页。
3. 修改“选择强名称密钥文件”属性。

可以使用 `AssemblyOriginatorKeyFile` 以编程方式访问此编译器选项。

请参见

[其他资源](#)

[C# 编译器选项](#)

/langversion(符合规范的语法)(C# 编译器选项)

导致编译器只接受 ISO/IEC 23270:2003 C# 语言规范中包含的语法。

```
/langversion:option
```

参数

option

如果 *option* 为 **ISO-1**, 则对于未包含在 ISO/IEC 23270:2003 C# 语言规范中的任何语法, 编译器都将报告错误。如果 *option* 为 **default**, 则编译器将接受所有有效的语法规法。**/langversion:default** 是默认值。

备注

C# 规范的 1.0 版表示 **/langversion:ISO-1** 中可用的功

能。<http://msdn.microsoft.com/vcsharp/programming/language/default.aspx> 包含 Microsoft Word 文件形式的所有规范。

C# 应用程序所引用的元数据不受 **/langversion** 编译器选项的影响。

C# 编译器的每个版本都包含语言规范的扩展, 因此 **/langversion** 不提供该编译器早期版本的等效功能。

无论您使用的是何种 **/langversion** 设置, 都将使用当前版本的公共语言运行库来创建 .exe 或 .dll。

这种情况的一个例外是友元程序集和 **/moduleassemblyname(指定模块的友元程序集)**(C# 编译器选项), 它们用于 **/langversion:ISO-1**。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“语言版本”属性。

有关如何以编程方式设置此编译器选项的信息, 请参见[LanguageVersion](#)。

请参见

[其他资源](#)

[C# 编译器选项](#)

/lib(指定程序集引用位置)(C# 编译器选项)

/lib 选项指定通过 [/reference\(导入元数据\)\(C# 编译器选项\)](#) 选项引用的程序集的位置。

```
/lib:dir1[,dir2]
```

参数

dir1

在当前工作目录(调用编译器的目录)或公共语言运行库的系统目录中未找到引用的程序集时, 编译器将在其中进行查找的目录。

dir2

要在其中搜索程序集引用的一个或多个附加目录。用逗号分隔每个附加目录的名称, 中间不要有空格。

备注

编译器按以下顺序搜索未完全限定的程序集引用:

1. 当前工作目录。该目录为从其调用编译器的目录。
2. 公共语言运行库系统目录。
3. 由 **/lib** 指定的目录。
4. 由 LIB 环境变量指定的目录。

使用 **/reference** 指定程序集引用。

/lib 是累加的;每一次指定的值都追加到以前的值中。

另一种使用 **/lib** 的方法是, 将任何所需的程序集复制到工作目录中;这使您可以仅将程序集名称传递给 **/reference**。然后可以从工作目录中删除这些程序集。由于程序集清单中未指定依赖程序集的路径, 因此应用程序可以在目标计算机上启动, 然后查找并使用全局程序集缓存中的程序集。

编译器可以引用程序集并不表示公共语言运行库可以在运行时找到并加载程序集。有关运行库如何搜索引用的程序集的详细信息, 请参见[运行库如何定位程序集](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性页”对话框。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“引用路径”属性页。
3. 修改列表框的内容。

有关如何以编程方式设置此编译器选项的信息, 请参见[ReferencePath](#)。

示例

编译 t2.cs 以创建 .exe 文件。编译器将在工作目录和驱动器 C 上根目录中查找程序集引用。

```
csc /lib:c:\ /reference:t2.dll t2.cs
```

请参见

其他资源

[C# 编译器选项](#)

/linkresource(链接到 .NET Framework 资源)(C# 编译器选项)

创建一个指向输出文件中的 .NET Framework 资源的链接，资源文件没有放置在输出文件中，这与 [/resource](#) 选项相反，该选项确实在输出文件中嵌入资源文件。

```
/linkresource:filename[,identifier[,accessibility-modifier]]
```

参数

filename

要从程序集链接的 .NET Framework 资源文件。

identifier(可选)

资源的逻辑名称；用于加载此资源的名称。默认为文件的名称。

accessibility-modifier(可选)

资源的可访问性：公共或私有。默认为公共。

备注

默认情况下，如果链接的资源是用 C# 编译器创建的，其在程序集中就是公共的。若要使这些资源成为私有的，请将 **private** 指定为可访问性修饰符。不允许使用 **public** 或 **private** 以外的其他修饰符。

/linkresource 需要除 **/target:module** 选项之外的 **/target** 选项之一。

如果 *filename* 是由例如 [Resgen.exe](#) 创建或在开发环境中创建的 .NET Framework 资源文件，则可以通过 [System.Resources](#) 命名空间中的成员访问该文件。有关更多信息，请参见 [System.Resources.ResourceManager](#)。对于所有其他资源，请使用 [Assembly](#) 类中的 **GetManifestResource*** 方法在运行时访问资源。

在 *filename* 中指定的文件可以为任何格式。例如，您可能想将本机 DLL 设置为程序集的一部分，以便可将其安装到全局程序集缓存中，并且可从程序集中的托管代码访问它。下面的第二个示例演示如何执行此操作。您可以在程序集链接器中执行相同的操作。下面的第三个示例演示如何执行此操作。有关更多信息，请参见 [程序集链接器 \(Al.exe\)](#) 和 [使用程序集和全局程序集缓存](#)。

/linkres 是 **/linkresource** 的缩写形式。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

示例

编译 in.cs 并链接到资源文件 rf.resource：

```
csc /linkresource:rf.resource in.cs
```

将 A.cs 编译为 DLL，链接到本机 DLL N.dll，并将输出放置在全局程序集缓存 (GAC) 中。在此示例中，A.dll 和 N.dll 都驻留在 GAC 中。

```
csc /linkresource:N.dll /t:library A.cs
gacutil -i A.dll
```

此示例和前一个示例执行的是同样的操作，但使用的是程序集链接器选项。

```
csc /t:module A.cs
al /out:A.dll A.netmodule /link:N.dll
gacutil -i A.dll
```

请参见

参考

[程序集链接器 \(Al.exe\)](#)

[其他资源](#)

[C# 编译器选项](#)

[使用程序集和全局程序集缓存](#)

/main(指定 Main 方法的位置)(C# 编译器选项)

如果有多个类包含 **Main** 方法，此选项指定包含程序入口点的类。

```
/main:class
```

参数

class

包含 **Main** 方法的类型。

备注

如果编译包括多个具有 **Main** 方法的类型，可以指定哪个类型包含要用作程序入口点的 **Main** 方法。

此选项用在编译 .exe 文件时。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“启动对象”属性。

若要以编程方式设置此编译器选项，请参见[StartupObject](#)。

示例

编译 t2.cs 和 t3.cs，并指定将在 Test2 中找到 **Main** 方法：

```
csc t2.cs t3.cs /main:Test2
```

[请参见](#)

[其他资源](#)

[C# 编译器选项](#)

/moduleassemblyname(指定模块的友元程序集)(C# 编译器选项)

指定 .netmodule 可访问其非公共类型的程序集。

```
/moduleassemblyname:assembly_name
```

参数

assembly_name

.netmodule 可以访问其非公共类型的程序集的名称。

备注

生成 .netmodule 并满足以下条件时，应使用 **/moduleassemblyname**：

- .netmodule 需要具有访问现有程序集中非公共类型的权限。
- 您知道 .netmodule 将生成到的程序集的名称。
- 现有程序集已经授予 .netmodule 将生成到的程序集友元程序集访问权限。

有关生成 .netmodule 的更多信息，请参见 [/target:module\(创建模块以添加到程序集\)\(C# 编译器选项\)](#)。

有关友元程序集的更多信息，请参见[友元程序集\(C# 编程指南\)](#)。

此选项在开发环境中不可用；它仅在从命令行编译时才可用。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

示例

此示例生成具有私有类型的程序集，并且该程序集授予称为 csman_an_assembly 的程序集友元程序集访问权限。

```
// moduleassemblyname_1.cs
// compile with: /target:library
using System;
using System.Runtime.CompilerServices;

[assembly:InternalsVisibleTo ("csman_an_assembly")]

class An_Internal_Class
{
    public void Test()
    {
        Console.WriteLine("An_Internal_Class.Test called");
    }
}
```

此示例生成可访问程序集 moduleassemblyname_1.dll 中非公共类型的 .netmodule。知道该 .netmodule 将生成到称为 csman_an_assembly 的程序集后，我们就可以指定 **/moduleassemblyname**，允许 .netmodule 访问已授予 csman_an_assembly 友元程序集访问权限的程序集中的非公共类型。

```
// moduleassemblyname_2.cs
// compile with: /moduleassemblyname:csman_an_assembly /target:module /reference:moduleassemblyname_1.dll
class B {
    public void Test() {
        An_Internal_Class x = new An_Internal_Class();
        x.Test();
    }
}
```

此代码示例通过引用以前生成的程序集和 .netmodule, 生成程序集 csman_an_assembly。

```
// csman_an_assembly.cs
// compile with: /addmodule:moduleassemblyname_2.netmodule /reference:moduleassemblyname_1.
dll
class A {
    public static void Main() {
        B bb = new B();
        bb.Test();
    }
}
```

输出

An_Internal_Class.Test called

[请参见](#)

[其他资源](#)

[C# 编译器选项](#)

/noconfig(忽略 csc.rsp)(C# 编译器选项)

/noconfig 选项通知编译器不要使用 csc.rsp 文件进行编译，该文件与 csc.exe 文件位于同一目录中，并从该目录中加载。

```
/noconfig
```

备注

csc.rsp 文件引用 .NET Framework 附带的所有程序集。Visual Studio .NET 开发环境包括的实际引用具体取决于项目类型。

可以修改 csc.rsp 文件并指定其他编译器选项，而这些选项是那些应该包括在使用 csc.exe 的来自命令行的每次编译中的选项 (**/noconfig** 选项除外)。

编译器会保留上次传递给 **csc** 命令的选项。因此，命令行上的任何选项都会重写 csc.rsp 文件中同一选项的设置。

如果不希望编译器查找并使用 csc.rsp 文件中的设置，请指定 **/noconfig**。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

其他资源

[C# 编译器选项](#)

/nologo(取消显示版权标志信息)(C# 编译器选项)

/nologo 选项在编译器启动时取消显示登录版权标志，并在编译期间取消显示信息性消息。

```
/nologo
```

备注

此选项在开发环境中不可用；它仅在从命令行编译时才可用。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

其他资源

[C# 编译器选项](#)

/nostdlib(不导入标准库)(C# 编译器选项)

/nostdlib 禁止导入定义整个 System 命名空间的 msclib.dll。

```
/nostdlib[<U>+</U> | -]
```

备注

如果您希望定义或创建自己的 System 命名空间和对象, 请使用该选项。

如果不指定 **/nostdlib**, msclib.dll 将导入程序(与指定 **/nostdlib-** 相同)。指定 **/nostdlib** 与指定 **/nostdlib+** 的作用相同。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 单击“高级”按钮。
4. 修改“不引用 msclib.dll”属性。

有关如何以编程方式设置此编译器选项的信息, 请参见[NoStdLib](#)。

请参见

[其他资源](#)

[C# 编译器选项](#)

/nowarn(取消显示指定警告)(C# 编译器选项)

/nowarn 选项允许您禁止编译器显示一个或多个警告。用逗号分隔多个警告编号。

```
/nowarn:number1[,number2,...]
```

参数

number1, number2

您希望编译器取消的警告编号。

备注

只应指定警告标识符的数值部分。例如，若要取消 CS0028，可以指定 /nowarn:28。

编译器在无人参与的模式下忽略传递给 **/nowarn** 的警告编号，这些警告编号在先前版本中有效但已从编译器中移除。例如，CS0679 在 Visual Studio .NET 2002 中有效但后来被删除。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“取消警告”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[DelaySign](#)。

请参见

其他资源

[C# 编译器选项](#)

/optimize(启用/禁用优化)(C# 编译器选项)

/optimize 选项启用或禁用由编译器执行以使输出文件更小、更快和更有效的优化。

```
/optimize[+ | <U>-</U>]
```

备注

/optimize 选项还通知公共语言运行库在运行时优化代码。

默认情况下，将禁用优化。指定 **/optimize+** 以启用优化。

在生成要由程序集使用的模块时，请使用与该程序集的设置相同的 **/optimize** 设置。

/o 是 **/optimize** 的缩写形式。

可以组合 **/optimize** 和 **/debug** 选项。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“优化代码”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[Optimize](#)。

示例

编译 t2.cs 并启用编译器优化：

```
csc t2.cs /optimize
```

请参见

其他资源

[C# 编译器选项](#)

/out(设置输出文件名)(C# 编译器选项)

/out 选项指定输出文件的名称。

```
/out:filename
```

参数

filename

由编译器创建的输出文件的名称。

备注

在命令行中，可以为编译指定多个输出文件。编译器希望在 **/out** 选项之后找到一个或多个源代码文件。这样一来，所有的源代码文件都将被编译到 **/out** 选项所指定的那个输出文件中。

指定要创建的文件的完整名称和扩展名。

如果不指定输出文件的名称：

- .exe 文件将从包含 **Main** 方法的源代码文件中获取其名称。
- .dll 或 .netmodule 文件将从第一个源代码文件中获取其名称。

用于编译一个输出文件的源代码文件不能在相同的编译中用作另一个输出文件的编译。

当在命令行编译中产生多个输出文件时，请记住其中只有一个输出文件可以是程序集并且只有用 **/out** 隐式或显式指定的第一个输出文件才是程序集。

作为编译的一部分产生的任何模块都变成与编译中产生的所有程序集相关的文件。使用 [ildasm.exe](#) 浏览程序集清单以查看相关的文件。

为使 EXE 成为友元程序集的目标，需要 **/out** 编译器选项。有关更多信息，请参见[友元程序集\(C# 编程指南\)](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“程序集名称”属性。

若要以编程方式设置此编译器选项：`OutputFileName` 是只读属性，它由项目类型(exe、库等)和程序集名称的组合确定。

必须修改这两个属性中的一个或全部才能设置输出文件名。

示例

编译 t.cs 并创建输出文件 t.exe，同时又生成 t2.cs 并创建模块输出文件 mymodule.netmodule：

```
csc t.cs /out:mymodule.netmodule /target:module t2.cs
```

请参见

参考

[友元程序集\(C# 编程指南\)](#)

其他资源

[C# 编译器选项](#)

/pdb(指定调试符号文件)(C# 编译器选项)

/pdb 编译器选项指定调试符号文件的名称和位置。

```
/pdb:filename
```

参数

filename

调试符号文件的名称和位置。

备注

当您指定 [/debug\(发出调试信息\)\(C# 编译器选项\)](#) 时，编译器将在创建输出文件 (.exe 或 .dll) 的同一目录下创建一个与输出文件同名的 .pdb 文件。

/pdb 允许您为 .pdb 文件指定非默认的文件名和位置。

不能在 Visual Studio 开发环境中设置此编译器选项，也不能以编程方式对其进行更改。

示例

编译 t.cs 并创建一个名为 tt.pdb 的 .pdb 文件：

```
csc /debug /pdb:tt t.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/platform(指定输出平台)(C# 编译器选项)

指定哪个版本的公共语言运行库 (CLR) 可以运行程序集。

```
/platform:string
```

参数

string

x86、Itanium、x64 或 anycpu(默认值)。

备注

- x86 将程序集编译为由兼容 x86 的 32 位公共语言运行库运行。
- Itanium 将程序集编译为由采用 Itanium 处理器的计算机上的 64 位公共语言运行库运行。
- x64 将程序集编译为由支持 AMD64 或 EM64T 指令集的计算机上的 64 位公共语言运行库运行。
- anycpu(默认值)将程序集编译为在任意平台上运行。

在 64 位 Windows 操作系统上：

- 用 **/platform:x86** 编译的程序集将在 WOW64 下运行的 32 位 CLR 上执行。
- 用 **/platform:anycpu** 编译的可执行文件将在 64 位 CLR 上执行。
- 用 **/platform:anycpu** 编译的 DLL 将在与加载它的进程相同的 CLR 上执行。

有关开发在 Windows 64 位操作系统上运行的应用程序的更多信息，请参见 [64 位应用程序](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“目标平台”属性。

注意 **/platform** 在 Visual C# 速成版开发环境中不可用。

有关如何以编程方式设置此编译器选项的信息，请参见 [PlatformTarget](#)。

示例

下面的示例演示如何使用 **/platform** 选项来指定应用程序只应由用于 Itanium 的 64 位 Windows 操作系统上的 64 位 CLR 运行。

```
csc /platform:Itanium myItanium.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/recurse(在子目录中查找源文件)(C# 编译器选项)

/recurse 选项使您可以编译指定目录 (*dir*) 或项目目录的所有子目录中的源代码文件。

```
/recurse:[dir\]file
```

参数

dir(可选)

搜索开始的目录。如果未指定此目录，则搜索从项目目录开始。

file

要搜索的文件。允许使用通配符字符。

备注

/recurse 选项使您可以编译指定目录 (*dir*) 或项目目录的所有子目录中的源代码文件。

可以在文件名中使用通配符来编译项目目录中所有匹配的文件，而不需使用 /recurse。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

示例

编译当前目录中的所有 C# 文件：

```
csc *.cs
```

编译 dir1\dir2 目录及其任何子目录中的所有 C# 文件，并生成 dir2.dll：

```
csc /target:library /out:dir2.dll /recurse:dir1\dir2\*.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/reference(导入元数据)(C# 编译器选项)

/reference 选项导致编译器将指定文件中的 [public\(C# 参考\)](#) 类型信息导入到当前项目中，从而使您可以从指定的程序集文件引用元数据。

```
/reference:[alias=]filename  
/reference:filename
```

参数

filename

包含程序集清单的文件的名称。若要导入多个文件，请为每个文件包括一个单独的 **/reference** 选项。

alias

一个有效的 C# 标识符，表示将包含程序集中所有命名空间的根命名空间。

备注

若要从多个文件导入，请为每个文件包括一个 **/reference** 选项。

导入的文件必须包含一个清单；输出文件必须已使用 [/target:module\(创建模块以添加到程序集\)\(C# 编译器选项\)](#) 以外的 [/target\(指定输出文件格式\)\(C# 编译器选项\)](#) 选项之一编译过。

/r 是 **/reference** 的缩写形式。

使用 [/addmodule\(导入元数据\)\(C# 编译器选项\)](#) 从不包含程序集清单的输出文件导入元数据。

如果您引用了一个程序集（程序集 A），其本身又引用了另一个程序集（程序集 B），则在下列情况下需要引用程序集 B：

- 使用来自程序集 A 的类型继承自程序集 B 中的类型或实现程序集 B 中的接口。
- 调用具有程序集 B 中的返回类型或参数类型的字段、属性、事件或方法。

使用 [/lib\(指定程序集引用位置\)\(C# 编译器选项\)](#) 指定一个或多个程序集引用所在的目录。[/lib](#) 主题还讨论了编译器在哪些目录中搜索程序集。

为使编译器可以识别程序集（而不是模块）中的某个类型，需要强制解析此类型，这可以通过定义此类型的实例来完成。还有其他方法可为编译器解析程序集中的类型名称：例如，如果您从程序集中的类型继承，编译器就能识别类型名称。

有时必须从一个程序集内部引用同一组件的两个不同版本。为此，请在每个文件的 **/reference** 开关上使用 *alias* 子选项，以区分两个文件。此别名将用作组件名称的限定符，并解析为其中一个文件中的组件。

默认情况下使用 csc 响应 (.rsp) 文件，该文件引用常用的 .NET Framework 程序集。如果不希望编译器使用 csc.rsp，请使用 [/noconfig\(忽略 csc.rsp\)\(C# 编译器选项\)](#)。

有关更多信息，请参见“[添加引用](#)”对话框。

示例

此示例演示如何使用 [外部别名\(C# 参考\)](#) 功能。

编译源文件，并从先前已编译的 grid.dll 和 grid20.dll 中导入元数据。这两个 DLL 包含同一组件的不同版本，您将使用两个带有别名选项的 **/reference** 编译源文件。这两个选项如下所示：

```
/reference:GridV1=grid.dll 和 /reference:GridV2=grid20.dll
```

这将设置外部别名“GridV1”和“GridV2”，您将在程序中通过外部语句使用它们：

```
extern GridV1;  
extern GridV2;  
// Using statements go here.
```

完成此操作后，您可以通过在控件名称前添加 GridV1 前缀来引用 grid.dll 中的网格控件，如下所示：

```
GridV1::Grid
```

此外，您可以通过在控件名称前添加 GridV2 前缀来引用 grid20.dll 中的网格控件，如下所示：

GridV2::Grid

请参见

其他资源

[C# 编译器选项](#)

/resource(将资源文件嵌入输出文件中)(C# 编译器选项)

将指定的资源嵌入到输出文件中。

```
/resource:filename[,identifier[,accessibility-modifier]]
```

参数

filename

要在输出文件中嵌入的 .NET Framework 资源文件。

identifier(可选)

资源的逻辑名称;用于加载此资源的名称。默认为文件的名称。

accessibility-modifier(可选)

资源的可访问性:公共或私有。默认为公共。

备注

使用 [/linkresource](#) 将资源链接到程序集, 但不将资源文件放置在输出文件中。

默认情况下, 用 C# 编译器创建的资源在程序集中是公共的。若要使这些资源成为私有的, 请将 **private** 指定为可访问性修饰符。不允许使用 **public** 或 **private** 以外的可访问性。

如果 *filename* 是由例如 [Resgen.exe](#) 创建或在开发环境中创建的 .NET Framework 资源文件, 则可以使用 [System.Resources](#) 命名空间中的成员访问该文件(有关更多信息, 请参见 [System.Resources.ResourceManager](#))。对于所有其他资源, 请使用 [Assembly](#) 类中的 **GetManifestResource*** 方法在运行时访问资源。

/res 是 **/resource** 的缩写形式。

资源在输出文件中的顺序是由命令行上指定的顺序决定的。

在 Visual Studio 开发环境中设置此编译器选项

1. 将资源文件添加到项目中。
2. 选择要嵌入解决方案资源管理器中的文件。
3. 在“属性”窗口中, 选择文件的“生成操作”。
4. 将“生成操作”设置为“嵌入的资源”。

有关如何以编程方式设置此编译器选项的信息, 请参见 [BuildAction](#)。

示例

编译 `in.cs` 并附加资源文件 `rf.resource`:

```
csc /resource:rf.resource in.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/target(指定输出文件格式)(C# 编译器选项)

/target 编译器选项可以指定为以下四种形式之一：

[/target:exe](#)

创建 .exe 文件。

[/target:library](#)

创建代码库。

[/target:module](#)

创建模块。

[/target:winexe](#)

创建 Windows 程序。

如果不指定 /target:module, /target 会将 .NET Framework 程序集清单放入输出文件中。有关更多信息, 请参见[公共语言运行库中的程序集和全局属性](#)。

程序集清单放置在编译中的第一个 .exe 输出文件中, 如果没有 .exe 输出文件, 会放置在第一个 DLL 中。例如, 在以下的命令行中, 清单将放置在 1.exe 中:

```
csc /out:1.exe t1.cs /out:2.netmodule t2.cs
```

编译器每次编译只创建一个程序集清单。关于编译中所有文件的信息全放在程序集清单中。除用 /target:module 创建的文件之外, 所有输出文件都可以包含程序集清单。在命令行生成多个输出文件时, 只能创建一个程序集清单, 且必须放置在命令行上指定的第一个输出文件中。无论第一个输出文件是什么 (/target:exe、/target:winexe、/target:library 或 /target:module), 在同一编译中生成的任何其他输出文件都必须是模块 (/target:module)。

如果创建了一个程序集, 则可以用 [CLSCompliant](#) 属性指示全部或部分代码是符合 CLS 的。

```
// target_clscompliant.cs
[assembly:System.CLSCompliant(true)] // specify assembly compliance
[System.CLSCompliant(false)] // specify compliance for an element
public class TestClass
{
    public static void Main() {}
}
```

有关以编程方式设置此编译器选项的更多信息, 请参见[OutputType](#)。

请参见

其他资源

[C# 编译器选项](#)

/target:exe(创建控制台应用程序)(C# 编译器选项)

/target:exe 选项使编译器创建可执行 (EXE) 的控制台应用程序。

```
/target:exe
```

备注

默认情况下, **/target:exe** 选项有效。将创建带扩展名 .exe 的可执行文件。

使用 [/target:winexe](#) 创建可执行的 Windows 程序。

除非另外使用 [/out](#) 选项指定, 否则输出文件名采用包含 [Main](#) 方法的输入文件名。

在命令行上指定该选项时, 下一个 [/out](#) 或 [/target:module](#) 选项之前的所有文件都将用于创建 .exe 文件。

在编译到 .exe 文件中的源代码文件中要求有且仅有一个 **main** 方法。[/main](#) 编译器选项使您可以在代码中包含多个具有 **main** 方法的类的情况下, 指定包含 **main** 方法的类。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“输出类型”属性。

有关如何以编程方式设置此编译器选项的信息, 请参见[OutputType](#)。

示例

下列每个命令行都将编译 in.cs, 创建 in.exe:

```
csc /target:exe in.cs
csc in.cs
```

请参见

参考

[/target\(指定输出文件格式\)\(C# 编译器选项\)](#)

其他资源

[C# 编译器选项](#)

/target:library(创建代码库)(C# 编译器选项)

/target:library 选项使编译器创建一个动态链接库 (DLL) 而不是一个可执行文件 (EXE)。

```
/target:library
```

备注

DLL 创建后会带有 .dll 扩展名。

除非使用 [/out](#) 选项另外指定，否则输出文件的名称采用第一个输入文件的名称。

在命令行上指定该选项时，下一个 [/out](#) 或 [/target:module](#) 选项之前的所有文件都将用于创建 .dll 文件。

生成 .dll 文件时，不需要 [main](#) 方法。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“输出类型”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[OutputType](#)。

示例

编译 in.cs，创建 in.dll：

```
csc /target:library in.cs
```

请参见

参考

[/target\(指定输出文件格式\)\(C# 编译器选项\)](#)

其他资源

[C# 编译器选项](#)

/target:module(创建模块以添加到程序集)(C# 编译器选项)

此选项导致编译器不会生成程序集清单。

```
/target:module
```

备注

默认情况下，使用此选项编译时所创建的输出文件具有 .netmodule 扩展名。

没有程序集清单的文件无法由 .NET Framework 公共语言运行库加载。但可以通过 [/addmodule](#) 将这类文件并入程序集清单中。

如果在一次编译中创建了多个模块，一个模块中的 `internal` 类型可用于此编译中的其他模块。如果一个模块中的代码引用另一个模块中的 `internal` 类型，则两个模块都必须通过 [/addmodule](#) 合并到一个程序集清单中。

Visual Studio 开发环境不支持创建模块。

有关如何以编程方式设置此编译器选项的信息，请参见 [OutputType](#)。

示例

编译 `in.cs`，创建 `in.netmodule`：

```
csc /target:module in.cs
```

请参见

参考

[/target\(指定输出文件格式\)\(C# 编译器选项\)](#)

其他资源

[C# 编译器选项](#)

/target:winexe(创建 Windows 程序)(C# 编译器选项)

/target:winexe 选项使编译器创建可执行 (EXE) 的 Windows 程序。

```
/target:winexe
```

备注

将创建带扩展名 .exe 的可执行文件。Windows 程序是一种由 .NET Framework 库或使用 Win32 API 提供一个用户界面的程序。

使用 [/target:exe](#) 创建控制台应用程序。

除非另外使用 [/out](#) 选项指定, 否则输出文件名采用包含 **Main** 方法的输入文件名。

在命令行指定该选项时, 直至下一个 [/out](#) 或 [/target](#) 选项之前的所有文件都将用于创建 Windows 程序。

在编译到 .exe 文件中的源代码文件中要求有且仅有一个 **main** 方法。[/main](#) 选项使您可以在代码中包含多个具有 **main** 方法的类的情况下, 指定包含 **main** 方法的类。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“输出类型”属性。

有关如何以编程方式设置此编译器选项的信息, 请参见[OutputType](#)。

示例

将 in.cs 编译成 Windows 程序:

```
csc /target:winexe in.cs
```

请参见

参考

[/target\(指定输出文件格式\)\(C# 编译器选项\)](#)

其他资源

[C# 编译器选项](#)

/unsafe(启用不安全模式)(C# 编译器选项)

/unsafe 编译器选项允许对使用 [unsafe](#) 关键字的代码进行编译。

```
/unsafe
```

备注

有关不安全代码的更多信息，请参见[不安全代码和指针\(C# 编程指南\)](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 选中“允许不安全代码”复选框。

有关如何以编程方式设置此编译器选项的信息，请参见[AllowUnsafeBlocks](#)。

示例

编译不安全模式的 `in.cs`:

```
csc /unsafe in.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/utf8output(使用 UTF-8 显示编译器消息)(C# 编译器选项)

/utf8output 选项使用 UTF-8 编码显示编译器输出。

```
/utf8output
```

备注

在某些国际配置中，编译器输出无法在控制台上正确显示。在这些配置中，请使用 **/utf8output** 并将编译器输出重定向到文件。

此编译器选项在 Visual Studio 中不可用，且不能通过编程方式进行更改。

请参见

其他资源

[C# 编译器选项](#)

/warn(指定警告等级)(C# 编译器选项)

/warn 选项指定编译器要显示的警告等级。

```
/warn:option
```

参数

option

您希望为编译显示的警告等级：较小的数字只显示高严重度的警告；数字越大，显示的警告越多。有效值为 0 到 4：

警告等级	含义
0	关闭所有警告消息的显示。
1	显示严重的警告消息。
2	显示等级 1 警告以及某些不太严重的警告，如关于隐藏类成员的警告。
3	显示等级 2 警告以及某些不太严重的警告，例如有关总是计算为 true 或 false 的表达式的警告。
4(默认)	显示所有等级 3 警告以及信息性警告。

备注

若要获得有关错误或警告的信息，可以在帮助索引中查找错误代码。有关获得错误或警告信息的其他方式，请参见[如何：定位编译器错误的帮助](#)。

使用 [/warnaserror](#) 可将所有的警告都视为错误。可以使用 [/nowarn](#) 禁用某些警告。

/w 是 **/warn** 的缩写形式。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“警告等级”属性。

有关如何以编程方式设置此编译器选项的信息，请参见[WarningLevel](#)。

示例

编译 `in.cs` 并让编译器仅显示等级 1 警告：

```
csc /warn:1 in.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/warnaserror(将警告视为错误)(C# 编译器选项)

/warnaserror+ 选项将所有警告都视为错误

```
/warnaserror[<U>+</U> | - ][:warning-list]
```

备注

将一般报告为警告的任何消息都报告为错误，并且暂停生成过程(不生成输出文件)。

默认情况下启用 **/warnaserror-**，这导致警告不会妨碍生成输出文件。**/warnaserror** 与 **/warnaserror+** 相同，它使警告被视为错误。

(可选)如果您希望只将几个特定的警告视为错误，可以指定一个以逗号分隔的列表，其中列出被视为错误的警告编号。

使用 **/warn** 可指定您希望编译器显示的警告等级。可以使用 **/nowarn** 禁用某些警告。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息，请参见[如何：设置项目属性\(C#、J#\)](#)。
2. 单击“生成”属性页。
3. 修改“将警告视为错误”属性。

若要以编程方式设置此编译器选项，请参见[TreatWarningsAsErrors](#)。

示例

编译 `in.cs` 并且不让编译器显示警告：

```
csc /warnaserror in.cs
csc /warnaserror:642,649,652 in.cs
```

请参见

其他资源

[C# 编译器选项](#)

/win32icon(导入.ico文件)(C# 编译器选项)

/win32icon 选项在输出文件中插入 .ico 文件, .ico 文件赋予输出文件在 Windows 资源管理器中的所需外观。

```
/win32icon:filename
```

参数

filename

要添加到输出文件的 .ico 文件。

备注

.ico 文件可以用[资源编译器](#)创建。编译 Visual C++ 程序时将调用资源编译器; .ico 文件是从 .rc 文件创建的。

请参见 [/linkresource](#)(用于引用 .NET Framework 资源文件)或 [/resource](#)(用于附加 .NET Framework 资源文件)。要导入 .res 文件, 请参见 [/win32res](#)。

在 Visual Studio 开发环境中设置此编译器选项

1. 打开项目的“属性”页。有关详细信息, 请参见[如何: 设置项目属性\(C#、J#\)](#)。
2. 单击“应用程序”属性页。
3. 修改“应用程序图标”属性。

有关如何以编程方式设置此编译器选项的信息, 请参见[ApplicationIcon](#)。

示例

编译 in.cs, 并附加 rf.ico 以生成 in.exe:

```
csc /win32icon:rf.ico in.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

/win32res(导入 Win32 资源文件)(C# 编译器选项)

/win32res 选项在输出文件中插入 Win32 资源。

```
/win32res:filename
```

参数

filename

要添加到输出文件的资源文件。

备注

Win32 资源文件可以用[资源编译器](#)创建。在编译 Visual C++ 程序时会调用资源编译器;.res 文件是用.rc 文件创建的。

Win32 资源可以包含版本或位图(图标)信息, 这些信息有助于在 Windows 资源管理器中标识您的应用程序。如果不指定 /win32res, 编译器将根据程序集版本生成版本信息。

请参见 [/linkresource](#)(用于引用 .NET Framework 资源文件)或 [/resource](#)(用于附加 .NET Framework 资源文件)。

此编译器选项在 Visual Studio 中不可用, 且不能通过编程方式进行更改。

示例

编译 in.cs, 并附加 Win32 资源文件 rf.res 以生成 in.exe:

```
csc /win32res:rf.res in.cs
```

请参见

[其他资源](#)

[C# 编译器选项](#)

如何：定位编译器错误的帮助

注意

显示的对话框和菜单命令可能会与“帮助”中的描述不同，具体取决于您的当前设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

所有的 C# 编译器错误都有相应的主题解释了错误生成的原因，在某些情况下还解释了如何修复错误。若要获取有关特定错误信息的帮助，请尝试以下操作之一：

过程

查找错误的帮助

- 单击“[输出](#)”窗口中的错误号，再按 F1。

- 或 -

在“索引”中的“查找”框中键入错误号。

- 或 -

在“搜索”页中键入错误号。

请参见

其他资源

[C# 编译器选项](#)

C# 编译器错误

本节包含按数字顺序列出的 C# 编译器错误。所有的 C# 编译器错误都有相应的主题解释了错误生成的原因，在某些情况下还解释了如何修复错误。若要获取有关特定错误信息的帮助，请尝试以下操作之一：

- 单击“[输出](#)”窗口中的错误号，再按 F1。
- 在“索引”中的“查找”框中键入错误号。
- 在“搜索”页中键入错误号。
- 在“目录”列表中找到该错误。

注意

显示的对话框和菜单命令可能会与“帮助”中描述的不同，具体取决于您现用的设置或版本。若要更改设置，请在“工具”菜单上选择“导入和导出设置”。有关更多信息，请参见 [Visual Studio 设置](#)。

请参见

其他资源

[C# 编译器选项](#)

编译器错误 CS0001

错误消息

内部编译器错误

尝试确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0003

错误消息

内存不足

编译器无法分配足够的虚拟内存完成编译。关闭所有不必要的应用程序并重新编译。

可能还需要增加页文件的大小, 请确保有可用的磁盘空间。

当 .NET Framework SDK 与 C# 编译器的版本不匹配时, 或者当一个或多个支持 C# 编译器的文件损坏时, 也可能出现此错误; 请重新安装 Visual Studio。

编译器错误 CS0004

视为错误的警告

您是用 [/warnaserror](#) 编译器选项编译的, 这将导致编译器生成被视为错误的警告。

编译器错误 CS0005

错误消息

编译器选项“compiler_option”后面必须有参数

有些编译器选项需要参数。如果不传递编译器选项所需的参数，则会生成 CS0005。

有关更多信息，请参见 [C# 编译器选项](#)。

编译器错误 CS0006

错误消息

未能找到元数据文件“dll_name”

编译了程序并给它显式传递了包含元数据的文件的名称，但没有找到 .dll。有关更多信息，请参见
[/reference\(导入元数据\)\(C# 编译器选项\)](#)。

编译器错误 CS0007

错误消息

意外的公共语言运行库初始化错误 —“description”

如果未能加载运行库，则会发生此错误。如果计算机上不存在编译器试图加载的公共语言运行库的版本，或者公共语言运行库的安装或配置已损坏，则可能发生此错误。

如果更改了 `csc.exe.config` 文件，则可能发生上述情况。此文件是在安装期间配置的，不应更改。如果 `csc.exe.config` 文件有可能已被更改，请检查该文件，确保计算机上存在该文件中指定的运行库的版本。如果存在正确的版本，它可能已损坏。请重新安装公共语言运行库。

编译器错误 CS0008

错误消息

从文件“file”读取元数据时发生错误 —“description”

成功打开了 DLL 以进行元数据检索，但它已损坏，因此未能从中读取数据。有关更多信息，请参见[/reference\(导入元数据\)\(C# 编译器选项\)](#)。

编译器错误 CS0009

错误消息

元数据文件“file”无法打开 — “description”

用 [/reference](#) 编译器选项指定的文件不包含有效的元数据。

编译器错误 CS0011

错误消息

无法解析由类型“type”引用的程序集“assembly”中的基类或接口“class”

用 **/reference** 从文件中导入的类是从某个类派生的，或者找不到它实现的接口。如果没有使用 **/reference** 将所需的 DLL 也包括在编译中，则可能发生这种情况。

有关更多信息，请参见[“添加引用”对话框](#)和[/reference\(导入元数据\)\(C# 编译器选项\)](#)。

示例

```
// CS0011_1.cs
// compile with: /target:library

public class Outer
{
    public class B { }
}
```

第二个文件创建一个 DLL，它定义从前一个示例中创建的类 B 派生的类 C。

```
// CS0011_2.cs
// compile with: /target:library /reference:CS0011_1.dll
// post-build command: del /f CS0011_1.dll
public class C : Outer.B {}
```

第三个文件替换第一步中创建的 DLL，并省略了内部类 B 的定义。

```
// CS0011_3.cs
// compile with: /target:library /out:cs0011_1.dll
public class Outer {}
```

最后，第四个文件引用在第二个示例中定义的类 C，该类从类 B 派生，但现在却丢失了。

下面的示例生成 CS0011。

```
// CS0011_4.cs
// compile with: /reference:CS0011_1.dll /reference:CS0011_2.dll
// CS0011 expected

class M
{
    public static void Main()
    {
        C c = new C();
    }
}
```

请参见

参考

[“添加引用”对话框](#)

[/reference\(导入元数据\)\(C# 编译器选项\)](#)

编译器错误 CS0012

错误消息

类型“type”在未被引用的程序集中定义。必须添加对程序集“assembly”的引用。

未找到引用类型的定义。如果所需的 .DLL 文件没有包括在编译中，则可能发生这种情况。有关更多信息，请参见[“添加引用”对话框和 /reference\(导入元数据\)\(C# 编译器选项\)](#)。

下面的编译序列将导致 CS0012:

```
// cs0012a.cs
// compile with: /target:library
public class A {}
```

然后:

```
// cs0012b.cs
// compile with: /target:library /reference:cs0012a.dll
public class B
{
    public static A f()
    {
        return new A();
    }
}
```

然后:

```
// cs0012c.cs
// compile with: /reference:cs0012b.dll
class C
{
    public static void Main()
    {
        object o = B.f();    // CS0012
    }
}
```

可以通过使用 `/reference:b.dll;a.dll` 进行编译来解决此 CS0012 错误。

编译器错误 CS0013

错误消息

将元数据写入文件“file”时发生错误 —“description”

.NET Framework 公共语言运行库未能发出元数据。请检查以确保路径正确并且磁盘未满。如果问题仍然存在，可能需要修复或重新安装 Visual Studio 和/或 .NET Framework。

编译器错误 CS0014

错误消息

无法找到所需的文件“file”

编译器需要某个文件，但它不在系统上。请确保路径是正确的。如果该文件是 Visual Studio 系统文件，那么您可能需要修复安装，或者移除并彻底重新安装 Visual Studio。

编译器错误 CS0015

错误消息

类型“type”的名称太长

用户定义类型的完全限定名必须少于或等于 2048 个字符。

下面的示例生成 CS0015:

编译器错误 CS0016

错误消息

未能写入输出文件“file”—“reason”

编译器未能写入输出文件。检查文件的路径，确保文件存在。如果该位置已经存在一个以前生成的文件，请确保该文件可写入，并且当前没有进程将该文件锁定。例如，确保在尝试生成时可执行文件没有运行。

编译器错误 CS0017

错误消息

程序“output file name”定义了不止一个入口点: function

程序只能有一个 `Main` 方法。

若要解决该错误, 可以删除代码中的所有 `Main` 方法, 只保留一个, 或者可以使用 `/main` 编译器选项指定要使用的 `Main` 方法。

下面的示例生成 CS0017:

```
// CS0017.cs
public class clx
{
    static public void Main()
    {
    }
}

public class cly
{
    public static void Main()    // CS0017, delete one Main or use /main
    {
    }
}
```

编译器错误 CS0019

错误消息

运算符“operator”无法应用在“type”和“type”类型的操作数

二进制运算符所操作的数据类型是该运算符不能处理的数据类型。例如，不能对字符串使用 `||` 运算符。

示例

在本例中，必须在 `ConditionalAttribute` 外指定条件逻辑。只能向 `ConditionalAttribute` 传递一个预定义符号。

下面的示例生成 CS0019。

```
// CS0019.cs
// compile with: /target:library
using System.Diagnostics;
public class MyClass
{
    [ConditionalAttribute("DEBUG" || "TRACE")]    // CS0019
    public void TestMethod() {}

    // OK
    [ConditionalAttribute("DEBUG")]
    public void TestMethod2() {}
}
```

请参见

参考

[运算符 \(C# 编程指南\)](#)

编译器错误 CS0020

错误消息

被常数零除

表达式在除法运算的分母中使用了字面(不是变量)值零。未定义被零除, 因此无效。

下面的示例生成 CS0020:

```
// CS0020.cs
namespace x
{
    public class b
    {
        public static void Main()
        {
            1 / 0;    // CS0020
        }
    }
}
```

请参见

参考

[运算符 \(C# 编程指南\)](#)

编译器错误 CS0021

错误消息

无法将带 [] 的索引应用于“type”类型的表达式

试图通过索引器访问不支持索引器([C# 编程指南](#))的数据类型的值。

当您试图在 C++ 程序集中使用索引器时，可能会遇到 CS0021。在这种情况下，请用 **DefaultMember** 属性修饰 C++ 类，以使 C# 编译器知道哪个索引器是默认的。下面的示例生成 CS0021。

示例

此文件编译成一个 .dll 文件(**DefaultMember** 属性被注释掉)以生成此错误。

```
// CPP0021.cpp
// compile with: /clr /LD
using namespace System::Reflection;
// Uncomment the following line to resolve
//[DefaultMember("myItem")]
public ref class MyClassMC
{
    public:
    property int myItem[int]
    {
        int get(int i){ return 5; }
        void set(int i, int value) {}
    }
};
```

下面是调用此 .dll 文件的 C# 文件。此文件试图通过索引器访问该类，但由于没有成员被声明为要使用的默认索引器，所以生成错误。

```
// CS0021.cs
// compile with: /reference:CPP0021.dll
public class MyClass
{
    public static void Main()
    {
        MyClassMC myMC = new MyClassMC();
        int j = myMC[1]; // CS0021
    }
}
```

编译器错误 CS0022

错误消息

[] 内索引数错误, 应为"number"

数组访问操作在方括号内指定的维数不正确。有关更多信息, 请参见[数组\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0022:

```
// CS0022.cs
public class MyClass
{
    public static void Main()
    {
        int[] a = new int[10];
        a[0] = 0;      // single-dimension array
        a[0,1] = 9;   // CS0022, the array does not have two dimensions
    }
}
```

编译器错误 CS0023

错误消息

运算符“operator”无法应用于“type”类型的操作数

试图将运算符应用于类型没有被设计用于该运算符的变量。有关更多信息, 请参见[数据类型\(C# 编程指南\)](#)和[C# 运算符](#)。

下面的示例生成 CS0023:

```
// CS0023.cs
namespace x
{
    public class a
    {
        public static void Main()
        {
            string s = "hello";
            s = -s;    // CS0023, minus operator not allowed on strings
        }
    }
}
```

编译器错误 CS0026

错误消息

关键字“this”在静态属性、静态方法或静态字段初始值设定项中无效

[this \(C# 参考\)](#) 关键字引用的对象是类型的实例。由于静态方法不依赖于包含类的任何实例，因此“this”关键字无意义，而这是不允许的。有关更多信息，请参见[静态类和静态类成员 \(C# 编程指南\)](#)和[对象 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0026：

```
// CS0026.cs
public class A
{
    public static int i = 0;

    public static void Main()
    {
// CS0026
        this.i = this.i + 1;
        // Try the following line instead:
        // i = i + 1;
    }
}
```

编译器错误 CS0027

错误消息

关键字“this”在当前上下文中不可用

在属性、方法或构造函数的外部发现了 [this\(C# 参考\)](#) 关键字。

若要修复此错误，请修改语句以消除 **this** 关键字的使用，并且/或者将语句的一部分或全部移到属性、方法或构造函数的内部。

下面的示例生成 CS0027：

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication3
{
    class MyClass
    {

        int err1 = this.Fun() + 1; // CS0027

        public int Fun()
        {
            return 10;
        }

        public void Test()
        {
            // valid use of this
            int err = this.Fun() + 1;
            Console.WriteLine(err);
        }

        public static void Main()
        {
            MyClass c = new MyClass();
            c.Test();
        }
    }
}
```

编译器错误 CS0029

错误消息

无法将类型“type”隐式转换为“type”

编译器要求显式转换。例如，可能需要将右值转换成与左值相同的类型。或者必须提供转换例程以支持某些运算符重载。

在将某个类型的变量赋给其他类型的变量时必须进行转换。当在不同类型的变量之间进行赋值时，编译器必须将赋值运算符右边的类型转换为该赋值运算符左边的类型。假设有下面的代码：

```
int i = 50;
long lng = 100;
i = lng;
```

`i = lng;` 进行赋值运算，但赋值运算符左右两边变量的数据类型不匹配。进行赋值前，编译器将变量 `lng`（类型为 `long`）隐式转换为 `int`。此为隐式转换，原因是没有代码显式指示编译器执行此转换。此代码的问题在于上述转换被视为收缩转换，而编译器不允许进行隐式收缩转换，原因是可能会丢失数据。

如果转换后的数据类型所占用的内存存储空间比转换前的数据类型所占用的少，则存在收缩转换。例如，将 `long` 类型转换为 `int` 类型就视为收缩转换。`long` 类型占用 8 个字节的内存，而 `int` 类型只占用 4 个字节。若要查看数据丢失如何发生，请考虑以下示例：

```
int i = 50;
long lng = 3147483647;
i = lng;
```

变量 `lng` 现在包含的值无法存储在变量 `i` 中，原因是该值太大。如果要将该值转换为 `int` 类型，就会丢失一些数据，并且转换后的值不同于转换前的值。

扩大转换与收缩转换相反。对于扩大转换，转换后的数据类型占用的内存存储空间比转换前的数据类型占用的多。下面是一个扩大转换的示例：

```
int i = 50;
long lng = 100;
lng = i;
```

请注意此代码示例和第一个示例之间的区别。这里，变量 `lng` 位于赋值运算符的左边，所以它是赋值的目标。在可以进行赋值前，编译器必须将变量 `i`（类型为 `int`）隐式转换为 `long` 类型，这是一个扩大转换，原因是从占用 4 个字节内存的类型 (`int`) 转换为占用 8 个字节内存的类型 (`long`)。由于不会发生数据丢失，所以允许进行隐式扩大转换。任何可以用 `int` 类型存储的值也可以用 `long` 类型存储。

我们知道隐式收缩转换是不允许的，因此为了能够编译这些代码，就需要显式转换数据类型。显式转换是使用强制转换来完成的。强制转换是 C# 中用来描述将一种数据类型转换为另一种数据类型的术语。若要编译这些代码，我们需要使用以下语法：

```
int i = 50;
long lng = 100;
i = (int) lng; // cast to int
```

第三行代码通知编译器在进行赋值前，将变量 `lng`（类型为 `long`）显式转换为 `int` 类型。切记，使用收缩转换可能会丢失数据。使用收缩转换时应小心，而且即使可以编译代码，也可能在运行时获得意外的结果。

此讨论只针对值类型。使用值类型就是直接使用存储在变量中的数据。但 .NET Framework 还具有引用类型。使用引用类型就是使用对变量的引用，而不是使用实际数据。引用类型的示例是类、接口和数组。不能隐式或显式地将一个引用类型转换为其他引用类型，除非编译器允许特定的转换或可以实现相应的转换运算符。

下面的示例生成 CS0029：

```
// CS0029.cs
public class MyInt
{
    private int x = 0;
```

```
// Uncomment this conversion routine to resolve CS0029
/*
public static implicit operator int(MyInt i)
{
    return i.x;
}
*/

public static void Main()
{
    MyInt myInt = new MyInt();
    int i = myInt; // CS0029
}
```

请参见

概念

[转换运算符 \(C# 编程指南\)](#)

编译器错误 CS0030

错误消息

无法将类型“type”转换为“type”

必须提供转换例程以支持某些运算符重载。有关更多信息, 请参见[转换运算符 \(C# 编程指南\)](#)。

下面的示例生成 CS0030:

```
// CS0030.cs
namespace x
{
    public class iii
    {
        /*
        public static implicit operator iii(int aa)
        {
            return null;
        }

        public static implicit operator int(iii aa)
        {
            return 0;
        }
    }

    public static iii operator ++(iii aa)
    {
        return (iii)0;    // CS0030
        // uncomment the conversion routines to resolve CS0030
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0031

错误消息

常数值“value”无法转换为“type”

试图对类型不能存储值的变量赋值。有关更多信息, 请参见[数据类型\(C# 编程指南\)](#)。

下面的示例生成 CS0031:

```
// CS0031.cs
namespace x
{
    public class a
    {
        public static void Main()
        {
            byte i = 512;    // CS0031, 512 cannot fit in byte
        }
    }
}
```

编译器错误 CS0034

错误消息

运算符“operator”对于“type1”和“type2”类型的操作数具有二义性

运算符用在了两个对象上，并且编译器找到多个转换。转换必须是唯一的，因此需要再进行一项类型转换，或者使其中一个转换成为显式的。有关更多信息，请参见[转换运算符\(C# 编程指南\)](#)。

下面的示例生成 CS0034：

```
// CS0034.cs
public class A
{
    // allows for the conversion of A object to int
    public static implicit operator int (A s)
    {
        return 0;
    }

    public static implicit operator string (A i)
    {
        return null;
    }
}

public class B
{
    public static implicit operator int (B s)
    // one way to resolve this CS0034 is to make one conversion explicit
    // public static explicit operator int (B s)
    {
        return 0;
    }

    public static implicit operator string (B i)
    {
        return null;
    }

    public static implicit operator B (string i)
    {
        return null;
    }

    public static implicit operator B (int i)
    {
        return null;
    }
}

public class C
{
    public static void Main ()
    {
        A a = new A();
        B b = new B();
        b = b + a;      // CS0034
        // another way to resolve this CS0034 is to make a cast
        // b = b + (int)a;
    }
}
```

编译器错误 CS0035

错误消息

运算符“operator”对于“type”类型的操作数具有二义性

编译器有多个可用的转换，但在应用运算符之前不知道选择哪个转换。有关更多信息，请参见[Templated User Defined Conversions](#) 和[转换运算符 \(C# 编程指南\)](#)。

下面的示例生成 CS0035：

```
// CS0035.cs
class MyClass
{
    private int i;

    public MyClass(int i)
    {
        this.i = i;
    }

    public static implicit operator double(MyClass x)
    {
        return (double) x.i;
    }

    public static implicit operator decimal(MyClass x)
    {
        return (decimal) x.i;
    }
}

class MyClass2
{
    static void Main()
    {
        MyClass x = new MyClass(7);
        object o = - x;    // CS0035
        // try a cast:
        // object o = - (double)x;
    }
}
```

编译器错误 CS0036

错误消息

输出参数不能具有 “[In]” 属性。

目前, [out](#) 参数上不允许有 **In** 属性。

下面的示例生成 CS0036:

```
// CS0036.cs

using System;
using System.Runtime.InteropServices;

public class MyClass
{
    public static void TestOut([In] out char TestChar) // CS0036
    // try the following line instead
    // public static void TestOut(out char TestChar)
    {
        TestChar = 'b';
        Console.WriteLine(TestChar);
    }

    public static void Main()
    {
        char i;           //variable to receive the value
        TestOut(out i); // the arg must be passed as out
        Console.WriteLine(i);
    }
}
```

编译器错误 CS0037

错误消息

无法将 NULL 转换成“type”，因为它是一个值类型

编译器不能给值类型赋空值，只能给引用类型或可为空值的类型赋空值。而 `struct` 是一个值类型。有关更多信息，请参见[可空类型\(C# 编程指南\)](#)。

下面的示例生成 CS0037：

```
// CS0037.cs
public struct s
{
}

class a
{
    public static void Main()
    {
        int i = null;    // CS0037
        s ss = null;    // CS0037
    }
}
```

编译器错误 CS0038

错误消息

无法通过嵌套类型“type2”来访问外部类型“type1”的非静态成员

类中的字段不自动可用于嵌套类。若要可用于嵌套类，该字段必须是 [static](#)。否则，必须创建外部类的实例。有关更多信息，请参见[嵌套类型 \(C# 编程指南\)](#)。

下面的示例生成 CS0038：

```
// CS0038.cs
class OuterClass
{
    public int count;
    // try the following line instead
    // public static int count;

    class InnerClass
    {
        void func()
        {
            // or, create an instance
            // OuterClass class_inst = new OuterClass();
            // int count2 = class_inst.count;
            int count2 = count;    // CS0038
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0039

错误消息

无法通过内置转换将类型“type1”转换为“type2”

继承、引用转换和装箱转换允许使用 [as\(C# 参考\)](#) 运算符的转换。有关更多信息，请参见 [转换运算符\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0039。

```
// CS0039.cs
using System;
class A
{
}
class B: A
{
}
class C: A
{
}
class M
{
    static void Main()
    {
        A a = new C();
        B b = new B();
        C c;

        // This is valid; there is a built-in reference
        // conversion from A to C.
        c = a as C;

        //The following generates CS0039; there is no
        // built-in reference conversion from B to C.
        c = b as C; // CS0039
    }
}
```

编译器错误 CS0040

错误消息

意外的调试信息初始化错误 –“reason”

在使用 [/debug](#) 编译器选项时会发生该错误，它指示编译器无法写入 .pdb 文件。该错误的可能解决方案包括重新安装 Visual Studio，确保编译器对文件或目录有写访问，或者不使用 /debug 进行编译。

编译器错误 CS0041

错误消息

将调试信息写入文件“file”时发生错误 —“reason”

在使用 [/debug](#) 编译器选项时可能会发生该错误。如果您遇到此错误，请尝试删除 bin 目录中的 PDB 文件并重新编译。如果仍然遇到此错误，可能需要修复或重新安装 Visual Studio。作为最后的选择，请尝试从 Microsoft 产品支持服务获得帮助 ([Visual Studio](#))。

编译器错误 CS0042

错误消息

创建调试信息文件“file”时发生错误 —“reason”

编译器未能创建调试信息; *reason* 包含错误条件的附加信息。

编译器错误 CS0043

错误消息

PDB 文件“file”的格式不正确或已过期。请删除该文件并重新生成。

删除用于此编译的 .pdb 文件并重新编译。

编译器错误 CS0050

错误消息

可访问性不一致: 返回类型“type”比方法“method”的可访问性低

方法的返回类型和形参表中引用的每个类型都必须至少具有和方法自身相同的可访问性。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0050, 因为没有为 `MyClass` 提供任何可访问性修饰符, 因此它的可访问性默认为 **private**。

```
// CS0050.cs
class MyClass //accessibility defaults to private
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public static MyClass MyMethod()    // CS0050
    {
        return new MyClass();
    }

    public static void Main() { }
}
```

编译器错误 CS0051

错误消息

可访问性不一致:参数类型“type”比方法“method”的访问性低

方法的返回类型和形参表中引用的每个类型都必须至少具有和方法自身相同的可访问性。请确保方法签名中使用的类型不会因为省略 **public** 修饰符而意外变为专用。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0051:

```
// CS0051.cs
public class A
{
    // Try making B public since F is public
    // B is implicitly private here
    class B
    {
    }

    public static void F(B b) // CS0051
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0052

错误消息

可访问性不一致: 字段类型“type”比字段“field”的访问性低

字段类型的可访问性不能比字段本身的可访问性低, 因为所有的公共构造都必须返回公共的可访问对象。

示例

下面的示例生成 CS0052:

```
// CS0052.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public MyClass mf;    // CS0052
}

public class MyClass3
{
    public static void Main()
    {
    }
}
```

请参见

参考

[C# 关键字](#)

[访问修饰符 \(C# 参考\)](#)

[可访问性级别 \(C# 参考\)](#)

[修饰符 \(C# 参考\)](#)

编译器错误 CS0053

错误消息

可访问性不一致: 属性类型“type”比属性“property”的可访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0053:

```
// CS0053.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public MyClass myProperty // CS0053
    {
        get
        {
            return new MyClass();
        }
        set
        {
        }
    }
}

public class MyClass3
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0054

错误消息

可访问性不一致: 索引器返回类型“type”比索引器“indexer”的可访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0054:

```
// CS0054.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass3
{
    public MyClass this[int i]    // CS0054
    {
        get
        {
            return new MyClass();
        }
    }
}

public class MyClass2
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0055

错误消息

可访问性不一致: 参数类型“type”比索引器“indexer”的访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0055:

```
// CS0055.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public int this[MyClass myClass]    // CS0055
    {
        get
        {
            return 0;
        }
    }
}

public class MyClass3
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0056

错误消息

可访问性不一致: 返回类型“type”比运算符“operator”的访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0056:

```
// CS0056.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class A
{
    public static implicit operator MyClass(A a) // CS0056
    {
        return new MyClass();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0057

错误消息

可访问性不一致: 参数类型“type”比运算符“operator”的访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0057:

```
// CS0057.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public class MyClass2
{
    public static implicit operator MyClass2(MyClass iii)    // CS0057
    {
        return new MyClass2();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0058

错误消息

可访问性不一致: 返回类型“type”比委托“delegate”的访问性低

公共构造必须返回可以公开访问的对象。有关更多信息, 请参见[访问修饰符\(C# 编程指南\)](#)。

下面的示例生成 CS0058, 因为没有访问修饰符应用于 MyClass, 因此默认情况下它的可访问性为 private:

```
// CS0058.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public delegate MyClass MyClassDel(); // CS0058

public class A
{
    public static void Main()
    {
    }
}
```

请参见

参考

[private\(C# 参考\)](#)

编译器错误 CS0059

错误消息

可访问性不一致:参数类型“type”比委托“delegate”的可访问性低

方法的返回类型和形参表中引用的每个类型都必须至少具有和方法自身相同的可访问性。有关更多信息,请参见[访问修饰符\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0059:

```
// CS0059.cs
class MyClass //defaults to private accessibility
// try the following line instead
// public class MyClass
{
}

public delegate void MyClassDel( MyClass myClass);    // CS0059

public class Program
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0060

错误消息

可访问性不一致: 基类“class1”比类“class2”的访问性低

基类和继承类之间的类可访问性应一致。

下面的示例生成 CS0060:

```
// CS0060.cs
class MyClass
// try the following line instead
// public class MyClass
{
}

public class MyClass2 : MyClass // CS0060
{
    public static void Main()
    {
    }
}
```

请参见

参考

[访问修饰符 \(C# 编程指南\)](#)

编译器错误 CS0061

错误消息

可访问性不一致: 基接口“interface 1”比接口“interface 2”的可访问性低

公共构造必须返回可以公开访问的对象。

在派生接口中, 接口可访问性不能被限定。有关更多信息, 请参见[接口 \(C# 编程指南\)](#)和[访问修饰符 \(C# 编程指南\)](#)。

下面的示例生成 CS0061。

```
// CS0061.cs
// compile with: /target:library
internal interface A {}
public interface AA : A {} // CS0061

// OK
public interface B {}
internal interface BB : B {}

internal interface C {}
internal interface CC : C {}
```

编译器错误 CS0065

错误消息

"event": 事件属性必须同时具有 add 访问器和 remove 访问器

不是字段的事件必须具有两种访问方法。

下面的示例生成 CS0065:

```
// CS0065.cs
using System;
public delegate void Eventhandler(object sender, int e);
public class MyClass
{
    public event EventHandler Click // CS0065,
    {
        // to fix, uncomment the add and remove definitions
        /*
        add
        {
            Click += value;
        }
        remove
        {
            Click -= value;
        }
        */
    }

    public static void Main()
    {
    }
}
```

请参见

概念

[事件 \(C# 编程指南\)](#)

编译器错误 CS0066

错误消息

"event": 事件必须是委托类型

event 关键字需要委托类型。有关更多信息, 请参见[事件\(C# 编程指南\)](#)和[委托\(C# 编程指南\)](#)。

下面的示例生成 CS0066:

```
// CS0066.cs
using System;

public class EventHandler
{
}

// to fix the error, remove the event declaration and the
// EventHandler class declaration, and uncomment the following line
// public delegate void EventHandler();

public class a
{
    public event EventHandler Click; // CS0066

    private void TestMethod()
    {
        if (Click != null)
            Click();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0068

错误消息

"event": 接口中的事件不能有初始值设定项

接口中的事件不能有初始值设定项。有关更多信息, 请参见[接口 \(C# 编程指南\)](#)。

下面的示例生成 CS0068:

```
// CS0068.cs

delegate void MyDelegate();

interface I
{
    event MyDelegate d = new MyDelegate(M.f);    // CS0068
    // try the following line instead
    // event MyDelegate d2;
}

class M
{
    event MyDelegate d = new MyDelegate(M.f);

    public static void f()
    {

    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0069

错误消息

接口中的事件不能具有 add 或 remove 访问器

不能在 [interface](#) 中定义事件的访问函数。有关更多信息, 请参见[事件 \(C# 编程指南\)](#)和[接口 \(C# 编程指南\)](#)。

下面的示例生成 CS0069:

```
// CS0069.cs
// compile with: /target:library

public delegate void EventHandler();

public interface A
{
    event EventHandler Click { remove {} }    // CS0069
    event EventHandler Click2;      // OK
}
```

编译器错误 CS0070

错误消息

事件“event”只能出现在 += 或 -= 的左边(从类型“type”中使用时除外)

在定义事件的类的外部, `event` 只能加上或减去引用。有关更多信息, 请参见[事件\(C# 编程指南\)](#)。

下面的示例生成 CS0070:

```
// CS0070.cs
using System;
public delegate void EventHandler();

public class A
{
    public event EventHandler Click;

    public static void OnClick()
    {
        EventHandler eh;
        A a = new A();
        eh = a.Click;
    }

    public static void Main()
    {
    }
}

public class B
{
    public int Foo ()
    {
        EventHandler eh = new EventHandler(A.OnClick);
        A a = new A();
        eh = a.Click;    // CS0070
        // try the following line instead
        // a.Click += eh;
        return 1;
    }
}
```

编译器错误 CS0071

错误消息

事件的显式接口实现必须使用属性语法

当显式实现在接口中声明的**事件**时，您必须手动提供通常由编译器提供的**add** 和 **remove** 事件访问器。访问器代码可将类中的接口事件连接到另一事件(如下所示)，或连接到其自身的委托类型。有关更多信息，请参见[如何: 实现接口事件\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0071。

```
// CS0071.cs
public delegate void MyEvent(object sender);

interface ITest
{
    event MyEvent Clicked;
}

class Test : Itest
{
    event MyEvent ITest.Clicked; // CS0071

    // try the following code instead
/*
private MyEvent clicked;

    event MyEvent Itest.Clicked
    {
        add
        {
            clicked += value;
        }
        remove
        {
            clicked -= value;
        }
    }
*/
    public static void Main() { }
```

编译器错误 CS0072

错误消息

"event":无法重写;"method"不是事件

[event](#) 只能重写另一事件。有关更多信息, 请参见[事件\(C# 编程指南\)](#)。

下面的示例生成 CS0072:

```
// CS0072.cs
delegate void MyDelegate();

class Test1
{
    public virtual event MyDelegate MyEvent;
    public virtual void VMeth()
    {
    }

    public void FireAway()
    {
        if (MyEvent != null)
            MyEvent();
    }
}

class Test2 : Test1
{
    public override event MyDelegate VMeth // CS0072
    // uncomment the following lines to resolve
    // public override event MyDelegate MyEvent
    {
        add
        {
            VMeth += value;
            // MyEvent += value;
        }
        remove
        {
            VMeth -= value;
            // MyEvent -= value;
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0073

错误消息

add 访问器或 remove 访问器必须有一个主体

`event` 定义中的 **add** 或 **remove** 关键字必须有体。有关更多信息, 请参见[事件 \(C# 编程指南\)](#)。

下面的示例生成 CS0073:

```
// CS0073.cs
delegate void del();

class Test
{
    public event del MyEvent
    {
        add;      // CS0073
        // try the following lines instead
        // add
        // {
        //     MyEvent += value;
        // }
        remove
        {
            MyEvent -= value;
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0074

错误消息

"event": 抽象事件不能有初始值设定项

如果 [event](#) 被标记为 **abstract**, 它就不能初始化。有关更多信息, 请参见[事件 \(C# 编程指南\)](#)。

下面的示例生成 CS0074:

```
// CS0074.cs
delegate void D();

abstract class Test
{
    public abstract event D e = null;    // CS0074
    // try the following line instead
    // public abstract event D e;

    public static void Main()
    {
    }
}
```

编译器错误 CS0075

错误消息

若要强制转换某个负值，必须将该值放在括号内

如果您使用标识预定义类型的关键字进行强制转换，则不需要括号。否则，必须用括号括起来，因为 $(x)-y$ 不会被视为强制转换表达式。摘自 C# 规范的第 7.6.6 节：

从消除歧义规则可以导出下述结论：如果 x 和 y 是标识符，则 $(x)y$ 、 $(x)(y)$ 和 $(x)(-y)$ 是强制转换表达式，但 $(x)-y$ 不是，即使 x 标识的是类型。但是，如果 x 是一个标识预定义类型（如 `int`）的关键字，则所有这四种形式均为强制转换表达式（因为这种关键字本身不可能是表达式）。

下面的代码生成 CS0075：

```
// CS0075
namespace MyNamespace
{
    enum MyEnum { }
    public class MyClass
    {
        public static void Main()
        {
            // To fix the error, place the negative
            // values below in parentheses
            int i = (System.Int32) - 4; //CS0075
            MyEnum e = (MyEnum) - 1; //CS0075
            System.Console.WriteLine(i); //to avoid warning
            System.Console.WriteLine(e); //to avoid warning
        }
    }
}
```

请参见

参考

[强制转换 \(C# 编程指南\)](#)

编译器错误 CS0076

错误消息

枚举名“value_”是保留字，无法使用

枚举中的项不能有称为 **value_** 的标识符。

编译器错误 CS0077

错误消息

as 运算符必须与引用类型一起使用 ("type" 是值类型)

向 as 运算符传递了值类型。由于 as 可以返回 null，因此只能向它传递引用类型。

下面的示例生成 CS0077：

```
// CS0077.cs
using System;

class C
{
}

struct S
{
}

class M
{
    public static void Main()
    {
        object o1, o2;
        C c;
        S s;

        o1 = new C();
        o2 = new S();

        s = o2 as S; // CS0077, S is not a reference type.
        // try the following line instead
        // c = o1 as C;
    }
}
```

编译器错误 CS0079

错误消息

事件“event”只能出现在 += 或 -= 的左边

未正确调用 [event](#)。有关更多信息, 请参见[事件\(C# 编程指南\)](#)和[委托\(C# 编程指南\)](#)。

下面的示例生成 CS0079:

```
// CS0079.cs
using System;

public delegate void MyEventHandler();

public class Class1
{
    private MyEventHandler _e;

    public event MyEventHandler Pow
    {
        add
        {
            _e += value;
            Console.WriteLine("in add accessor");
        }
        remove
        {
            _e -= value;
            Console.WriteLine("in remove accessor");
        }
    }

    public void Handler()
    {
    }

    public void Fire()
    {
        if (_e != null)
        {
            Pow(); // CS0079
            // try the following line instead
            // _e();
        }
    }

    public static void Main()
    {
        Class1 p = new Class1();
        p.Pow += new MyEventHandler(p.Handler);
        p._e();
        p.Pow += new MyEventHandler(p.Handler);
        p._e();
        p._e -= new MyEventHandler(p.Handler);
        if (p._e != null)
        {
            p._e();
        }
        p.Pow -= new MyEventHandler(p.Handler);
        if (p._e != null)
        {
            p._e();
        }
    }
}
```


编译器错误 CS0080

错误消息

在非泛型声明上不允许使用约束

找到的语法只能用于泛型声明才能对类型参数应用约束。有关更多信息, 请参见[泛型\(C# 编程指南\)](#)。

下面的示例生成 CS0080, 因为 MyClass 不是泛型类, 并且 Foo 不是泛型方法。

```
namespace MyNamespace
{
    public class MyClass where MyClass : System.IDisposable // CS0080      //the following line shows an example of correct syntax
    //public class MyClass<T> where T : System.IDisposable
    {
        public void Foo() where Foo : new() // CS0080
        //the following line shows an example of correct syntax
        //public void Foo<U>() where U : struct
        {
        }
    }

    public class Program
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0081

错误消息

类型参数声明必须是标识符而不是类型

当您声明泛型方法或类型时, 请将类型参数指定为标识符, 例如“T”或“inputType”。客户端代码在调用方法时会提供类型, 以替换方法体或类体中标识符的每个匹配项。有关更多信息, 请参见[泛型类型参数\(C# 编程指南\)](#)。

```
// CS0081.cs
class MyClass
{
    public void F<int>() {}    // CS0081
    public void F<T>(T input) {} // OK

    public static void Main()
    {
        MyClass a = new MyClass();
        a.F<int>(2);
        a.F<double>(.05);
    }
}
```

请参见

概念

[泛型\(C# 编程指南\)](#)

编译器错误 CS0100

错误消息

参数名“parameter name”重复

方法声明多次使用了同一个参数名。参数名在方法声明中必须唯一。有关更多信息, 请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0100:

```
// CS0100.cs
namespace x
{
    public class a
    {
        public static void f(int i, char i)    // CS0100
        // try the following line instead
        // public static void f(int i, char j)
        {
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0101

错误消息

命名空间“namespace”已经包含了“type”的定义

[namespace](#) 有重复的标识符。重命名或删除其中一个重复的标识符。有关更多信息, 请参见[命名空间 \(C# 编程指南\)](#)

下面的示例生成 CS0101:

```
// CS0101.cs
namespace MyNamespace
{
    public class MyClass
    {
        static public void Main()
        {
        }
    }

    public class MyClass // CS0101
    {
    }
}
```

编译器错误 CS0102

错误消息

类型“type name”已经包含“identifier”的定义

一个类在同一范围内包含多个具有相同名称的标识符的声明。若要修复此错误，请重命名重复的标识符。

示例

下面的示例生成 CS0102。

```
// CS0102.cs
// compile with: /target:library
namespace MyApp
{
    public class MyClass
    {
        string s = "Hello";
        string s = "Goodbye"; // CS0102

        public void GetString()
        {
            string s = "Hello again"; // method scope, no error
        }
    }
}
```

编译器错误 CS0103

错误消息

当前上下文中不存在名称“identifier”

试图使用类、[命名空间](#)或范围中不存在的名称。检查名称的拼写，并检查 using 语句和程序集引用，确保您尝试使用的名称可用。一个常见的错误是在一个循环或 try 块内声明一个变量，然后试图从封闭代码块或其他代码块中访问该变量，如下面的示例所示。

下面的示例生成 CS0103：

```
// CS0103.cs
using System;

class MyClass
{
    public static void Main()
    {
        // MyClass conn = null;
        try
        {
            MyClass conn = new MyClass();    // delete this line
            // and uncomment the following line and the line above the try
            // conn = new MyClass();
        }
        catch(Exception e)
        {
            if (conn != null)    // CS0103
                Console.WriteLine("{0}", e);
        }
    }
}
```

编译器错误 CS0104

错误消息

"reference"是"identifier"和"identifier"之间不明确的引用

程序包含两个命名空间的 `using` 指令, 而且代码引用了在这两个命名空间中均出现的名称。

下面的示例生成 CS0104:

```
// CS0104.cs
using x;
using y;

namespace x
{
    public class Test
    {
    }
}

namespace y
{
    public class Test
    {
    }
}

public class a
{
    public static void Main()
    {
        Test test = new Test();    // CS0104, is Test in x or y namespace?
        // try the following line instead
        // y.Test test = new y.Test();
    }
}
```

编译器错误 CS0106

错误消息

修饰符“modifier”对该项无效

类或接口成员是用无效的访问修饰符标记的。下列示例描述了一些无效的修饰符：

- 在接口方法上不允许使用 **static** 和 **public** 修饰符。
- 在显式接口声明上不允许使用 **public** 关键字。在这种情况下，请从显式接口声明中移除 **public** 关键字。
- 在显式接口声明上不允许使用 **abstract** 关键字，因为显式接口实现永远不能被重写。

在以前的 Visual Studio 版本中，不允许在类上使用 **static** 修饰符，但允许 **static** 类以 Microsoft Visual Studio 2005 开头。

有关更多信息，请参见[接口 \(C# 编程指南\)](#)

示例

下面的示例生成 CS0106。

```
// CS0106.cs
namespace MyNamespace
{
    interface I
    {
        void m();
        static public void f();    // CS0106
    }

    public class MyClass
    {
        public void I.m() {}    // CS0106
        public static void Main() {}
    }
}
```

编译器错误 CS0107

错误消息

不止一个保护修饰符

类成员上只允许使用一个访问修饰符([public](#)、[private](#)、[protected](#) 或 [internal](#))，**internal protected** 是个例外。

下面的示例生成 CS0107:

```
// CS0107.cs
public class C
{
    private internal void f()    // CS0107, delete private or internal
    {
    }

    public static int Main()
    {
        return 1;
    }
}
```

编译器错误 CS0110

错误消息

"const declaration"的常数值计算涉及循环定义

`const` 变量 (a) 的声明不能引用另一个也引用 (a) 的 `const` 变量 (b)。

下面的示例生成 CS0110:

```
// CS0110.cs
namespace MyNamespace
{
    public class A
    {
        public static void Main()
        {
        }
    }

    public class B : A
    {
        public const int i = c + 1;    // CS0110, c already references i
        public const int c = i + 1;
        // the following line would be OK
        // public const int c = 10;
    }
}
```

请参见

参考

[常量 \(C# 编程指南\)](#)

编译器错误 CS0111

错误消息

类型“class”已经定义了一个具有相同参数类型的名为“member”的成员

如果一个类包含两个具有相同名称和参数类型的成员声明，则会发生 CS0111。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0111。

```
// CS0111.cs
class A
{
    void Test() { }
    public static void Test(){ } // CS0111
    public static void Main() {}
}
```

编译器错误 CS0112

错误消息

静态成员“function”不能标记为 override、virtual 或 abstract

使用 **override**、**virtual** 或 **abstract** 关键字的任何方法声明不能还使用 **static** 关键字。

有关更多信息, 请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0112:

```
// CS0112.cs
namespace MyNamespace
{
    abstract public class MyClass
    {
        public abstract void Foo();
    }
    public class MyClass2 : MyClass
    {
        override public static void Foo() // CS0112, remove static keyword
        {
        }
        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0113

错误消息

标记为 `override` 的成员“function”不能标记为 `new` 或 `virtual`

用 `new` 和 `override` 关键字标记方法是互斥的。

下面的示例生成 CS0113:

```
// CS0113.cs
namespace MyNamespace
{
    abstract public class MyClass
    {
        public abstract void Foo();
    }

    public class MyClass2 : MyClass
    {
        override new public void Foo() // CS0113, remove new keyword
        {
        }

        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0115

错误消息

"function": 没有找到适合的方法来重写

方法被标记为 `override`, 但编译器未找到可重写的方法。有关更多信息, 请参见[override\(C# 参考\)](#)和[了解何时使用 Override 和 New 关键字\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0115。可以用下列两种方法之一解决 CS0115:

- 从 `MyClass2` 的方法中移除 `override` 关键字。
- 将 `MyClass1` 用作 `MyClass2` 的基类。

```
// CS0115.cs
namespace MyNamespace
{
    abstract public class MyClass1
    {
        public abstract int f();
    }

    abstract public class MyClass2
    {
        public override int f() // CS0115
        {
            return 0;
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0116

错误消息

命名空间并不直接包含诸如字段或方法之类的成员

在 [namespace](#) 内, 编译器只接受类、结构、联合、枚举、接口和委托。当 C/C++ 开发人员忘记了在 C# 中, 方法和变量必须在结构或类中定义时, 通常会生成此错误。有关更多信息, 请参见 [C# 程序的通用结构\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0116:

```
// CS0116.cs
namespace x
{
    using System;

    // method must be in class/struct
    void Method(string str) // CS0116
    {
        Console.WriteLine(str);
    }
}
```

编译器错误 CS0117

"type"不包含"identifier"的定义

当引用的成员对于此数据类型不存在时，将出现此错误。

几种常见的情况可产生此错误：

- 调用的方法不存在。
- 使用的 **Item** 属性后面跟有索引器。
- 当一个类的名称和它的封闭命名空间名称相同时，调用限定方法。
- 调用的接口是用支持接口内部的静态成员的语言编写的。

下面的示例生成 CS0117。

```
// CS0117_1.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i;
            i = i.get(); // CS0117
        }
    }
}
```

示例

在此示例中，**Item** 属性和索引器一起使用。在 C# 中，您可以使用属性或索引器访问成员，但不能同时使用这两者访问成员。下面的示例生成 CS0117。

```
// CS0117_2.cs
using System;
using System.Collections;
class Test
{
    public static void Main()
    {
        ArrayList al = new ArrayList();
        al.Add( new Test() );
        Console.WriteLine("{0}", al.Item[0]); // CS0117
        Console.WriteLine("{0}", al[0]); // OK
    }
}
```

如果您使用的库是用允许接口中存在静态成员的语言编写的，当您尝试从 C# 中访问静态成员时，也会发生 CS0017。

```
// CS0117_3.jsl
// compile with: /target:library
public interface IMyJSharpInterface
{
    static int MyStaticMember = 0;
    void NonStaticMember();
}
```

下面的示例生成 CS0117。

```
// CS0117_4.cs
```

```
// compile with: /reference:CS0117_3.dll
class MyCSharpClass : IMyJSharpInterface
{
    public void NonStaticMember() {}

    public static void Main()
    {
        IMyJSharpInterface myObj = new MyCSharpClass();
        myObj.NonStaticMember();
        int i = myObj.MyStaticMember;    // CS0117
    }
}
```

编译器错误 CS0118

错误消息

"construct1_name"是一个"construct1", 但其用法与"construct2"一样

编译器检测到以错误的方式使用了某个构造, 或对某个构造尝试了不允许的操作。一些常见示例包括:

- 尝试实例化命名空间(而不是类)
- 尝试调用字段(而不是方法)
- 尝试将类型作为变量
- 尝试使用外部别名作为类型。

若要解决此错误, 请确保您正在执行的操作对操作所针对的类型是有效的。

示例

下面的示例生成 CS0118。

```
// CS0118.cs
// compile with: /target:library
namespace MyNamespace
{
    class MyClass
    {
        // MyNamespace not a class
        MyNamespace ix = new MyNamespace ();    // CS0118
    }
}
```

编译器错误 CS0119

错误消息

"construct1_name"是一个"construct1", 它在给定的上下文中无效。

编译器检测到意外构造, 如下所示:

- 类构造函数在条件语句中是无效的测试表达式。
- 在引用数组元素时使用的是类名而不是实例名。
- 按结构或类那样使用方法标识符。

示例

下面的示例生成 CS0119。

```
// CS0119.cs
using System;
public class MyClass
{
    public static void Test() {}

    public static void Main()
    {
        Console.WriteLine(Test.x);    // CS0119
    }
}
```

编译器错误 CS0120

错误消息

非静态的字段、方法或属性“member”要求对象引用

必须首先创建对象实例，才能使用非静态的字段、方法或属性。有关更多信息，请参见[实例构造函数\(C# 编程指南\)](#)。

下面的示例生成 CS0120：

```
// CS0120_1.cs
public class MyClass
{
    // Non-static field
    public int i;
    // Non-static method
    public void f(){}
    // Non-static property
    int Prop
    {
        get
        {
            return 1;
        }
    }

    public static void Main()
    {
        i = 10;      // CS0120
        f();        // CS0120
        int p = Prop;    // CS0120
        // try the following lines instead
        // MyClass mc = new MyClass();
        // mc.i = 10;
        // mc.f();
        // int p = mc.Prop;
    }
}
```

如果从静态方法调用非静态方法，也会生成 CS0120，如下所示：

```
// CS0120_2.cs
// CS0120 expected
using System;

public class MyClass
{
    public static void Main()
    {
        TestCall();    // CS0120
        // To call a non-static method from Main,
        // first create an instance of the class.
        // Use the following two lines instead:
        // MyClass anInstanceOfMyClass = new MyClass();
        // anInstanceOfMyClass.TestCall();
    }

    public void TestCall()
    {
    }
}
```

同样，静态方法不能调用实例方法，除非显式给它提供了类的实例，如下所示：

```
// CS0120_3.cs
```

```
using System;

public class MyClass
{
    public static void Main()
    {
        do_it("Hello There"); // CS0120
    }

    private void do_it(string sText)
    // You could also add the keyword static to the method definition:
    // private static void do_it(string sText)
    {
        Console.WriteLine(sText);
    }
}
```

编译器错误 CS0121

错误消息

在以下方法或属性之间的调用不明确：“method1”和“method2”

因隐式转换的缘故，编译器无法调用重载方法的某种形式。可以用以下方法纠正该错误：

- 以不发生隐式转换的方式指定此方法的参数。
- 移除此方法的所有重载。
- 在调用方法之前，强制转换到正确的类型。

下面的示例生成 CS0121：

```
// CS0121.cs
public class C
{
    void f(int i, double d)
    {
    }

    void f(double d, int i)
    {
    }

    public static void Main()
    {
        C c = new C();

        c.f(1, 1);      // CS0121
        // try the following line instead
        // c.f(1, 1.0);
        // or
        // c.f(1.0, 1);
        // or
        // c.f(1, (double)1);    // cast and specify which method to call
    }
}
```

编译器错误 CS0122

错误消息

不可访问“member”，因为它受保护级别限制

类成员的[访问修饰符](#)禁止访问该成员。有关更多信息，请参见[访问修饰符\(C# 编程指南\)](#)。

出现此错误(未在下面的示例中显示)的一个原因是:在友元程序集的目标上省略了[/out](#)编译器标志。有关更多信息，请参见[友元程序集\(C# 编程指南\)](#)和[/out\(设置输出文件名\)\(C# 编译器选项\)](#)。

示例

下面的示例生成 CS0122:

```
// CS0122.cs
public class MyClass
{
    // Make public to resolve CS0122
    void Foo()
    {
    }
}

public class MyClass2
{
    public static int Main()
    {
        MyClass a = new MyClass();
        // Foo is private
        a.Foo();    // CS0122
        return 0;
    }
}
```

编译器错误 CS0123

错误消息

"method"的重载均与委托"delegate"不匹配

试图创建委托时失败，因为未使用正确的签名。必须使用与委托声明相同的签名声明委托实例。

可以通过调整方法或委托签名解决此错误。有关更多信息，请参见[委托\(C# 编程指南\)](#)。

下面的示例生成 CS0123。

```
// CS0123.cs
delegate void D();
delegate void D2(int i);

public class C
{
    public static void f(int i) {}

    public static void Main()
    {
        D d = new D(f);    // CS0123
        D2 d2 = new D2(f); // OK
    }
}
```

编译器错误 CS0126

错误消息

需要一个类型可转换为“type”的对象

存在返回语句，但该语句并未返回所需类型的值。有关更多信息，请参见[属性 \(C# 编程指南\)](#)。

下面的示例生成 CS0126：

```
// CS0126.cs
public class a
{
    public int i
    {
        set
        {
        }
        get
        {
            return; // CS0126, specify a value to return
        }
    }
}
```

编译器错误 CS0127

错误消息

由于“function”返回 void，返回关键字后面不得有对象表达式

具有 [void](#) 返回类型的方法无法返回值。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0127：

```
// CS0127.cs
namespace MyNamespace
{
    public class MyClass
    {
        public int hiddenMember2
        {
            get
            {
                return 0;
            }
            set // CS0127, set has an implicit void return type
            {
                return 0; // remove return statement to resolve this CS0127
            }
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0128

错误消息

已在此范围定义了名为“variable”的局部变量

编译器检测到两个同名的局部变量的声明。有关更多信息, 请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0128:

```
// CS0128.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            char i;
            int i;    // CS0128
        }
    }
}
```

编译器错误 CS0131

错误消息

赋值号左边必须是变量、属性或索引器

在赋值语句中，右边的值赋给左边。左边必须是变量、属性或索引器。

若要修复此错误，请确保所有运算符都位于右边，而左边是变量、属性或索引器。有关更多信息，请参见[语句、表达式和运算符 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0131。

```
// CS0131.cs
public class MyClass
{
    public int i = 0;
    public void MyMethod()
    {
        i++ = 1;      // CS0131
        // try the following line instead
        // i = 1;
    }
    public static void Main() { }
```

如果您试图对赋值运算符的左边执行算术运算(如下面的示例中所示)，也可能发生此错误。

```
// CS0131b.cs
public class C
{
    public static int Main()
    {
        int a = 1, b = 2, c = 3;
        if (a + b = c) // CS0131
        // try this instead
        // if (a + b == c)
            return 0;
        return 1;
    }
}
```

编译器错误 CS0132

错误消息

"constructor": 静态构造函数必须无参数

[static 构造函数不能用一个或多个参数声明。有关更多信息, 请参见构造函数\(C# 编程指南\)。](#)

下面的示例生成 CS0132:

```
// CS0132.cs
namespace MyNamespace
{
    public class MyClass
    {
        public MyClass(int i)
        {
        }
    }

    public class MyClass2 : MyClass
    {
        static MyClass2(int i) // CS0132
        {
        }
    }
}
```

编译器错误 CS0133

错误消息

指派给“variable”的表达式必须是常数

`const` 变量不能采用非常数的表达式作为它的值。有关更多信息, 请参见[常量 \(C# 编程指南\)](#)。

下面的示例生成 CS0133:

```
// CS0133.cs
public class MyClass
{
    public const int i = c;    // CS0133, c is not constant
    public static int c = i;
    // try the following line instead
    // public const int i = 6;

    public static void Main()
    {
    }
}
```

编译器错误 CS0134

错误消息

"variable"的类型为"type"。只能用 null 对引用类型(字符串除外)的常量进行初始化

常量表达式是可在编译时可完全计算的表达式。由于应用 new 运算符是创建引用类型的非空值的唯一方法，并且常量表达式中不允许使用 new 运算符，因此除字符串外，引用类型常量的唯一可能值为 null。

如果尝试创建[常量](#)字符串数组时遇到此错误，则解决方案是使该数组成为[只读](#)数组，并在构造函数中将其初始化。

示例

下面的示例生成 CS0134。

```
// CS0134.cs
// compile with: /target:library
class MyTest {}

class MyClass
{
    const MyTest test = new MyTest();      // CS0134

    //OK
    const MyTest test2 = null;
    const System.String test3 = "test";
}
```

编译器错误 CS0135

错误消息

"declaration1"与声明"declaration2"冲突

编译器不允许名称隐藏，名称隐藏通常会导致代码中的逻辑错误。

下面的示例生成 CS0135：

```
// CS0135.cs
public class MyClass2
{
    public static int i = 0;

    public static void Main()
    {
        {
            int i = 4;
            i++;
        }
        i = 0;    // CS0135
    }
}
```

编译器错误 CS0136

错误消息

不能在此范围内声明名为“var”的局部变量，因为这样会使“var”具有不同的含义，而它已经用于“parent or current/child”范围以表示其他内容

变量声明隐藏了本应在范围内的另一个声明。请重命名在生成 CS0136 的行上声明的变量。

下面的示例生成 CS0136：

```
// CS0136.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i = 0;
            {
                char i = 'a';    // CS0136, hides int i
            }
            i++;
        }
    }
}
```

编译器错误 CS0138

错误消息

using 命名空间指令只能应用于命名空间;“type”是一个类型, 而不是命名空间

using 指令只能将命名空间的名称用作参数。有关更多信息, 请参见[命名空间\(C# 编程指南\)](#)。

下面的示例生成 CS0138:

```
// CS0138.cs
using System.Object; // CS0138
```

编译器错误 CS0139

错误消息

没有要中断或继续的封闭循环

在循环外遇到了 break 或 continue 语句。

有关更多信息, 请参见[跳转语句](#)。

下面的示例生成 CS0139 两次:

```
// CS0139.cs
namespace x
{
    public class a
    {
        public static void Main()
        {
            continue; // CS0139
            break;    // CS0139
        }
    }
}
```

编译器错误 CS0140

错误消息

标签“label”重复

同名的标签出现了两次。有关更多信息, 请参见 [goto\(C# 参考\)](#)。

下面的示例生成 CS0140:

```
// CS0140.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            label1: int i = 0;
            label1: int j = 0;    // CS0140, comment this line to resolve
            goto label1;
        }
    }
}
```

编译器错误 CS0143

错误消息

类型“class”未定义构造函数

没有适当的构造函数可用。

下面的示例生成 CS0143:

```
// CS0143.cs
class MyClass
{
    static public void Main ()
    {
        double d = new double(4.5);    // CS0143
    }
}
```

编译器错误 CS0144

错误消息

无法创建抽象类或接口“interface”的实例

不能创建 `abstract` 类或 `interface` 的实例。有关更多信息, 请参见[接口 \(C# 编程指南\)](#)。

下面的示例生成 CS0144:

```
// CS0144.cs
interface MyInterface
{
}
public class MyClass
{
    public static void Main()
    {
        MyInterface myInterface = new MyInterface (); // CS0144
    }
}
```

编译器错误 CS0145

错误消息

常数段要求提供一个值

必须初始化 `const` 变量。有关更多信息, 请参见[常量 \(C# 编程指南\)](#)。

下面的示例生成 CS0145:

```
// CS0145.cs
class MyClass
{
    const int i;    // CS0145
    // try the following line instead
    // const int i = 0;

    public static void Main()
    {
    }
}
```

编译器错误 CS0146

错误消息

涉及“class1”和“class2”的循环基类依赖项

类的继承列表包括对自身的直接或间接引用。类不能从自身继承。有关更多信息，请参见[继承 \(C# 编程指南\)](#)。

下面的示例生成 CS0146:

```
// CS0146.cs
namespace MyNamespace
{
    public interface InterfaceA
    {

    }

    public class MyClass : InterfaceA, MyClass2
    {
        public void Main()
        {
        }
    }

    public class MyClass2 : MyClass // CS0146
    {
    }
}
```

编译器错误 CS0148

错误消息

委托“delegate”没有有效的构造函数

您导入并使用了用另一个编译器创建的托管程序(即使用 .NET Framework 公共语言运行库的托管程序)。该编译器允许了格式错误的 `delegate` 构造函数。有关更多信息, 请参见[委托\(C# 编程指南\)](#)。

编译器错误 CS0149

错误消息

应输入方法名称

在创建 `delegate` 时指定方法。有关更多信息, 请参见委托(C# 编程指南)。

下面的示例生成 CS0149:

```
// CS0149.cs
using System;

delegate string MyDelegate(int i);

class MyClass
{
    // class member-field of the declared delegate type
    static MyDelegate dt;

    public static void Main()
    {
        dt = new MyDelegate(17.45);    // CS0149
        // try the following line instead
        // dt = new MyDelegate(Func2);
        F(dt);
    }

    public static string Func2(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void F(MyDelegate myFunc)
    {
        myFunc(8);
    }
}
```

编译器错误 CS0150

错误消息

应输入常数值

在需要常数的地方发现了变量。有关更多信息, 请参见 [switch\(C# 参考\)](#)。

下面的示例生成 CS0150:

```
// CS0150.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i = 0;
            int j = 0;

            switch(i)
            {
                case j: // CS0150, j is a variable int, not a constant int
                // try the following line instead
                // case 1:
            }
        }
    }
}
```

编译器错误 CS0151

错误消息

应输入整型值

在需要整型数据类型的情况下使用了变量。有关更多信息, 请参见[数据类型\(C# 编程指南\)](#)。

示例

当没有转换或者可用的隐式转换导致了不明确的情形时, 会出现此错误。下面的示例生成 CS0151。

```
// CS0151.cs
public class MyClass
{
    public static implicit operator int (MyClass aa)
    {
        return 0;
    }

    public static implicit operator long (MyClass aa)
    {
        return 0;
    }

    public static void Main()
    {
        MyClass a = new MyClass();

        // Compiler cannot choose between int and long
        switch (a)    // CS0151
        // try the following line instead
        // switch ((int)a)
        {
            case 1:
                break;
        }
    }
}
```

编译器错误 CS0152

错误消息

标签“label”已经出现在此 switch 语句中

在 [switch](#) 语句中重复使用了某个标签。有关更多信息，请参见 [switch\(C# 参考\)](#)。

下面的示例生成 CS0152:

```
// CS0152.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            int i = 0;

            switch (i)
            {
                case 1:
                    i++;
                    return;

                case 1: // CS0152, two case 1 statements
                    i++;
                    return;
            }
        }
    }
}
```

编译器错误 CS0153

错误消息

goto case 只在 switch 语句中有效

试图在 **switch** 语句外部使用 **switch** 语法。有关更多信息, 请参见 [switch\(C# 参考\)](#)。

下面的示例生成 CS0153:

```
// CS0153.cs
public class a
{
    public static void Main()
    {
        goto case 5;    // CS0153
    }
}
```

编译器错误 CS0154

错误消息

属性或索引器“property”无法用于此上下文中，因为它缺少 get 访问器

使用属性的尝试失败，因为该属性中未定义 get 访问器方法。有关更多信息，请参见[字段 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0154：

```
// CS0154.cs
public class MyClass2
{
    public int i
    {
        set
        {
        }
        // uncomment the get method to resolve this error
        /*
        get
        {
            return 0;
        }
        */
    }
}

public class MyClass
{
    public static void Main()
    {
        MyClass2 myClass2 = new MyClass2();
        int j = myClass2.i;    // CS0154, no get method
    }
}
```

编译器错误 CS0155

错误消息

捕获或抛弃的类型必须从 System.Exception 派生

试图将不是从 **System.Exception** 派生的数据类型传递到 **catch** 块中。只有从 **System.Exception** 派生的数据类型才可以传递到 **catch** 块。有关更多信息, 请参见[异常处理语句](#)和[异常和异常处理\(C# 编程指南\)](#)。

下面的示例生成 CS0155:

```
// CS0155.cs
using System;

namespace MyNamespace
{
    public class MyClass2
    // try the following line instead
    // public class MyClass2 : Exception
    {
    }
    public class MyClass
    {
        public static void Main()
        {
            try
            {
            }
            catch (MyClass2) // CS0155, resolves if you derive MyClass2 from Exception
            {
            }
        }
    }
}
```

编译器错误 CS0156

错误消息

无参数的 throw 语句不允许在 catch 子句之外使用

不带参数的 `throw` 语句只能出现在不带参数的 `catch` 子句中。

有关更多信息, 请参见[异常处理语句](#)和[异常和异常处理\(C# 编程指南\)](#)。

下面的示例生成 CS0156:

```
// CS0156.cs
using System;

namespace MyNamespace
{
    public class MyClass2 : Exception
    {
    }

    public class MyClass
    {
        public static void Main()
        {
            try
            {
                throw;      // CS0156
            }

            catch(MyClass2)
            {
                throw;      // this throw is valid
            }
        }
    }
}
```

编译器错误 CS0157

错误消息

控制不能离开 finally 子句主体

`finally` 子句中的所有语句必须执行。有关更多信息, 请参见[异常处理语句和异常和异常处理\(C# 编程指南\)](#)。

下面的示例生成 CS0157:

```
// CS0157.cs
using System;
namespace MyNamespace
{
    public class MyClass2 : Exception
    {
    }

    public class MyClass
    {
        public static void Main()
        {
            try
            {

            }

            finally
            {
                return; // CS0157, cannot leave finally clause
            }
        }
    }
}
```

编译器错误 CS0158

错误消息

在包含的范围中标签“label”遮盖了具有同样名称的另一个标签

内部范围中的标签隐藏了外部范围中的同名标签。有关更多信息, 请参见 [goto\(C# 参考\)](#)。

下面的示例生成 CS0158:

```
// CS0158.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            goto lab1;
            lab1:
            {
                lab1:
                goto lab1;    // CS0158
            }
        }
    }
}
```

编译器错误 CS0159

错误消息

goto 语句范围内没有“label”这样的标签

在 **goto** 语句范围内无法找到 **goto** 语句所引用的标签。

下面的示例生成 CS0159:

```
// CS0159.cs
public class Class1
{
    public static void Main()
    {
        int i = 0;

        switch (i)
        {
            case 1:
                goto case 3;    // CS0159, case 3 label does not exist
            case 2:
                break;
        }
        goto NOWHERE;    // CS0159, NOWHERE label does not exist
    }
}
```

编译器错误 CS0160

错误消息

上一个 catch 子句已经捕获了此类型或超类型("type")的所有异常

一系列 **catch** 语句需要按派生的递减顺序排列。例如，派生程度最高的对象必须首先出现。

有关更多信息，请参见[异常处理语句](#)和[异常和异常处理\(C# 编程指南\)](#)。

下面的示例生成 CS0160：

```
// CS0160.cs
public class MyClass2 : System.Exception {}
public class MyClass
{
    public static void Main()
    {
        try {}

        catch(System.Exception) {}    // Second-most derived; should be second catch
        catch(MyClass2) {}    // CS0160  Most derived; should be first catch
    }
}
```

编译器错误 CS0161

错误消息

"method": 并非所有的代码路径都返回值

返回值的方法必须在所有代码路径中都具有 **return** 语句。有关更多信息, 请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0161:

```
// CS0161.cs
public class Test
{
    public static int Main() // CS0161
    {
        int i = 10;
        if (i < 10)
        {
            return i;
        }
        else
        {
            // uncomment the following line to resolve
            // return 1;
        }
    }
}
```

编译器错误 CS0163

错误消息

控制不能从一个 case 标签("label")贯穿到另一个 case 标签

当 **case** 语句包含一个或多个语句，并且后跟另一个 **case** 语句时，下面的语句之一必须显式地终止 **case** 语句：

- **return**
- **goto**
- **break**

如果要实现“贯穿”行为，请使用 `goto case #`。有关更多信息，请参见 [switch\(C# 参考\)](#)

下面的示例生成 CS0163：

```
// CS0163.cs
public class MyClass
{
    public static void Main()
    {
        int i = 0;

        switch (i) // CS0163
        {
            case 1:
                i++;
                // uncomment one of the following lines to resolve
                // return;
                // break;
                // goto case 3;

            case 2:
                i++;
                return;

            case 3:
                i = 0;
                return;
        }
    }
}
```

编译器错误 CS0165

错误消息

使用了未赋值的局部变量“var”

C# 编译器不允许使用未初始化的变量。如果编译器检测到使用了可能未初始化的变量，就会生成 CS0165。有关更多信息，请参见[字段\(C# 编程指南\)](#)。

使用 `new` 创建对象实例或赋值。

下面的示例生成 CS0165：

```
// CS0165.cs
using System;

class MyClass
{
    public int i;
}

class MyClass2
{
    public static void Main(string [] args)
    {
        int i, j;
        if (args[0] == "test")
        {
            i = 0;
        }

        /*
        // to resolve, either initialize the variables when declared
        // or provide for logic to initialize them, as follows:
        else
        {
            i = 1;
        }
        */

        j = i;      // CS0165, i might be uninitialized

        MyClass myClass;
        myClass.i = 0;      // CS0165
        // use new as follows
        // MyClass myClass = new MyClass();
        // myClass.i = 0;
    }
}
```

编译器错误 CS0167

错误消息

委托“delegate”缺少 Invoke 方法

您导入并使用了用另一个编译器创建的托管程序(即使用 .NET Framework 公共语言运行库的托管程序)。该编译器允许了格式错误的 `delegate`。因此, `Invoke` 方法不可用。有关更多信息, 请参见[委托\(C# 编程指南\)](#)。

编译器错误 CS0170

错误消息

使用了可能未赋值的字段“field”

结构中的字段在使用前未初始化。若要解决此问题，首先确定哪个字段尚未初始化，然后将其初始化，再尝试访问该字段。有关初始化结构的更多信息，请参见[结构\(C# 编程指南\)](#)和[使用结构\(C# 编程指南\)](#)。

下面的示例生成 CS0170:

编译器错误 CS0171

错误消息

在控制离开构造函数之前，字段“field”必须完全赋值

`struct` 中的构造函数必须初始化结构中的所有字段。有关更多信息，请参见[构造函数\(C# 编程指南\)](#)。

下面的示例生成 CS0171：

```
// CS0171.cs
struct MyStruct
{
    MyStruct(int initField) // CS0171
    {
        // i = initField;      // uncomment this line to resolve this error
    }
    public int i;
}

class MyClass
{
    public static void Main()
    {
        MyStruct aStruct = new MyStruct();
    }
}
```

编译器错误 CS0172

错误消息

无法确定条件表达式的类型，因为“type1”和“type2”可相互隐式转换

在条件语句中，: 运算符两边的类型必须可以相互转换。而且，不能有相互转换例程；只需一个转换。有关更多信息，请参见[转换运算符\(C# 编程指南\)](#)。

下面的示例生成 CS0172：

```
// CS0172.cs
public class Square
{
    public class Circle
    {
        public static implicit operator Circle(Square aa)
        {
            return null;
        }

        public static implicit operator Square(Circle aa)
        // using explicit resolves this error
        // public static explicit operator Square(Circle aa)
        {
            return null;
        }
    }

    public static void Main()
    {
        Circle aa = new Circle();
        Square ii = new Square();
        object o = (1 == 1) ? aa : ii;    // CS0172
        // the following cast would resolve this error
        // (1 == 1) ? aa : (Circle)ii;
    }
}
```

编译器错误 CS0173

错误消息

无法确定条件表达式的类型，因为“class1”和“class2”之间没有隐式转换

当希望不同类的对象在同一代码中使用时，类之间的转换非常有用。然而，在一起工作的两个类不能有相互转换和多余转换。

若要解决 CS0173，请确认 *class1* 和 *class2* 之间有且只有一个隐式转换，而不论在哪一个方向上转换或在哪一个类中转换。有关更多信息，请参见[隐式数值转换表\(C# 参考\)](#)和[转换运算符\(C# 编程指南\)](#)。

下面的示例生成 CS0173：

```
// CS0173.cs
public class C {}
public class A {}

public class MyClass
{
    public static void F(bool b)
    {
        A a = new A();
        C c = new C();
        object o = b ? a : c; // CS0173
    }

    public static void Main()
    {
        F(true);
    }
}
```

编译器错误 CS0174

错误消息

"base"引用需要基类

该错误仅在 .NET Framework 公共语言运行库编译 **System.Object** 类(它是唯一没有基类的类)的源代码时才发生。

编译器错误 CS0175

错误消息

在此上下文中使用关键字“base”无效

必须使用 [base\(C# 参考\)](#) 关键字指定基类的特定成员。有关更多信息, 请参见[构造函数\(C# 编程指南\)](#)。

下面的示例生成 CS0175:

```
// CS0175.cs
using System;
class BaseClass
{
    public int TestInt = 0;
}

class MyClass : BaseClass
{
    public static void Main()
    {
        MyClass aClass = new MyClass();
        aClass.BaseTest();
    }

    public void BaseTest()
    {
        Console.WriteLine(base); // CS0175
        // Try the following line instead:
        // Console.WriteLine(base.TestInt);
        base = 9; // CS0175

        // Try the following line instead:
        // base.TestInt = 9;
    }
}
```

编译器错误 CS0176

错误消息

无法使用实例引用访问静态成员“member”；改用类型名来限定它

只有类名称可用于限定 `static` 变量；实例名称不能是限定符。有关更多信息，请参见[静态类和静态类成员 \(C# 编程指南\)](#)。

下面的示例生成 CS0176：

```
// CS0176.cs
public class MyClass2
{
    public static int ii;
}

public class a
{
    public static void Main()
    {
        MyClass2 myClass2 = new MyClass2 ();
        int i = myClass2.ii;    // CS0176
        // try the following line instead
        // int i = MyClass2.ii;
    }
}
```

编译器错误 CS0177

错误消息

控制离开当前方法之前必须对输出参数“parameter”赋值

在方法体中没有给用 `out` 关键字标记的参数赋值。有关更多信息, 请参见[传递参数 \(C# 编程指南\)](#)

下面的示例生成 CS0177:

```
// CS0177.cs
public class MyClass
{
    public static void Foo(out int i) // CS0177
    {
        // uncomment the following line to resolve this error
        // i = 0;
    }

    public static void Main()
    {
        int x = -1;
        Foo(out x);
    }
}
```

编译器错误 CS0178

错误消息

无效的秩说明符: 应为","或"]"

数组初始化的格式错误。例如，在指定数组维数时，可以指定以下内容：

- 中括号中的数字。
- 空的中括号。
- 括在中括号中的逗号。

有关更多信息，请参见[数组\(C# 编程指南\)](#)和[C# 规范\(C# 语言规范\)](#)中有关数组初始值设定项的部分。

示例

下面的示例生成 CS0178。

```
// CS0178.cs
class MyClass
{
    public static void Main()
    {
        int a = new int[5][,][,][5];    // CS0178
        int[,] b = new int[3,2];        // OK

        int[][] c = new int[10][];
        c[0] = new int[5][5];          // CS0178
        c[0] = new int[2];             // OK
        c[1] = new int[2]{1,2};        // OK
    }
}
```

编译器错误 CS0179

错误消息

"member"不能是外部的, 也无法声明主体

当类成员被标记为 `extern` 时, 意味着该成员的定义位于另一文件中。因此, 标记为 `extern` 的类成员不能在类中定义。请移除 `extern` 关键字或删除定义。有关更多信息, 请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0179:

```
// CS0179.cs
public class MyClass
{
    public extern int ExternMethod(int aa)    // CS0179
    {
        return 0;
    }
    // try the following line instead
    // public extern int ExternMethod(int aa);

    public static void Main()
    {
    }
}
```

编译器错误 CS0180

错误消息

"member"不能既是外部的又是抽象的

abstract 和 **extern** 关键字是互斥的。**extern** 关键字表示该成员在文件外部定义，而 **abstract** 表示其实现在派生类中提供。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0180：

```
// CS0180.cs
namespace MyNamespace
{
    public class MyClass
    {
        public extern abstract int Foo(int a); // CS0180

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0182

错误消息

属性参数必须是常数表达式、typeof 表达式或数组创建表达式

未正确指定属性的参数。有关更多信息, 请参见[全局属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0182:

```
// CS0182.cs
public class MyClass
{
    static string s = "Test";

    [System.Diagnostics.ConditionalAttribute(s)] // CS0182
    // try the following line instead
    // [System.Diagnostics.ConditionalAttribute("Test")]
    void NonConstantArgumentToConditional()
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0185

错误消息

"type"不是 lock 语句要求的引用类型

lock 语句只能计算引用类型。有关更多信息, 请参见[线程同步 \(C# 编程指南\)](#)和[引用类型 \(C# 参考\)](#)。

示例

下面的示例生成 CS0185:

```
// CS0185.cs
public class MainClass
{
    public static void Main ()
    {
        lock (1)    // CS0185
        // try the following lines instead
        // MainClass x = new MainClass();
        // lock(x)
        {
        }
    }
}
```

编译器错误 CS0186

错误消息

在此上下文中使用 NULL 无效

下面的示例生成 CS0186:

```
// CS0186.cs
using System;
using System.Collections;

class MyClass
{
    static void Main()
    {
        // Each of the following lines generates CS0186:
        foreach (int i in null) {}    // CS0186
        foreach (int i in (IEnumerable) null) { };    // CS0186
    }
}
```

编译器错误 CS0188

错误消息

在给“this”对象的所有字段赋值之前，无法使用该对象

构造函数必须给 **struct** 中的所有字段都赋值，然后才能调用 **struct** 中的方法。有关更多信息，请参见[使用结构\(C# 编程指南\)](#)。

下面的示例生成 CS0188：

```
// CS0188.cs
// compile with: /t:library
namespace MyNamespace
{
    class MyClass
    {
        struct S
        {
            public int a;

            void Foo()
            {
            }

            S(int i)
            {
                // a = i;
                Foo(); // CS0188
            }
        }
        public static void Main()
        { }

    }
}
```

编译器错误 CS0191

错误消息

无法对只读的字段赋值(构造函数或变量初始值指定项中除外)

`readonly` 字段只能在构造函数或声明中获得赋值。有关更多信息, 请参见[构造函数 \(C# 编程指南\)](#)。

如果 `readonly` 字段为 `static`, 但构造函数没有标记为 `static`, 也会导致 CS0191。

示例

下面的示例生成 CS0191。

```
// CS0191.cs
class MyClass
{
    public readonly int TestInt = 6; // OK to assign to readonly field in declaration

    MyClass()
    {
        TestInt = 11; // OK to assign to readonly field in constructor
    }

    public void TestReadOnly()
    {
        TestInt = 19; // CS0191
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0192

错误消息

对只读字段无法传递 `ref` 或 `out` (构造函数中除外)

用 `readonly` 关键字标记的字段(变量)不能传递给 `ref` 或 `out` 参数, 除非在构造函数内。有关更多信息, 请参见[字段\(C# 编程指南\)](#)。

如果 `readonly` 字段为 `static` 而构造函数没有被标记为 `static`, 则也会导致 CS0192。

示例

下面的示例生成 CS0192。

```
// CS0192.cs
class MyClass
{
    public readonly int TestInt = 6;
    static void TestMethod(ref int testInt)
    {
        testInt = 0;
    }

    MyClass()
    {
        TestMethod(ref TestInt);    // OK
    }

    public void PassReadOnlyRef()
    {
        TestMethod(ref TestInt);    // CS0192
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0193

错误消息

* 或 -> 运算符只能应用于指针

* 或 -> 运算符用于了非指针类型。有关更多信息, 请参见[指针类型\(C# 编程指南\)](#)。

下面的示例生成 CS0193:

```
// CS0193.cs
using System;

public struct Age
{
    public int AgeYears;
    public int AgeMonths;
    public int AgeDays;
}

public class MyClass
{
    public static void SetAge(ref Age anAge, int years, int months, int days)
    {
        anAge->Months = 3;    // CS0193, anAge is not a pointer
        // try the following line instead
        // anAge.AgeMonths = 3;
    }

    public static void Main()
    {
        Age MyAge = new Age();
        Console.WriteLine(MyAge.AgeMonths);
        SetAge(ref MyAge, 22, 4, 15);
        Console.WriteLine(MyAge.AgeMonths);
    }
}
```

编译器错误 CS0196

错误消息

指针必须只根据一个值进行索引

指针不能有多个索引。有关更多信息, 请参见[指针类型\(C# 编程指南\)](#)。

下面的示例生成 CS0196:

```
// CS0196.cs
public class MyClass
{
    public static void Main ()
    {
        int *i = null;
        int j = 0;
        j = i[1,2];    // CS0196
        // try the following line instead
        // j = i[1];
    }
}
```

编译器警告(等级 1)CS0197

错误消息

由于“argument”是引用封送类的字段，将它作为 `ref` 或 `out` 参数传递或获取它的地址可能导致运行时异常。

从 [MarshalByRefObject](#) 派生(直接或间接)的任何类均是引用封送类。这样的类可以通过引用跨越进程和计算机边界进行封送。因此，该类的实例可以是远程对象的代理。不能将代理对象的字段作为 `ref` 或 `out` 传递。因此，不能将这种类的字段作为 `ref` 或 `out` 传递，除非实例是 `this`(它不能是代理对象)。

示例

下面的示例生成 CS0197。

```
// CS0197.cs
// compile with: /W:1
class X : System.MarshalByRefObject
{
    public int i;
}

class M
{
    public int i;
    static void AddSeventeen(ref int i)
    {
        i += 17;
    }

    static void Main()
    {
        X x = new X();
        x.i = 12;
        AddSeventeen(ref x.i);    // CS0197

        // OK
        M m = new M();
        m.i = 12;
        AddSeventeen(ref m.i);
    }
}
```

编译器错误 CS0198

错误消息

无法对静态只读字段赋值(静态构造函数或变量初始值设定项中除外)

`readonly` 变量与要在其中初始化它的构造函数必须具有相同的 `static` 用法。有关更多信息, 请参见 [静态构造函数\(C# 编程指南\)](#)。

下面的示例生成 CS0198:

```
// CS0198.cs
class MyClass
{
    public static readonly int TestInt = 6;

    MyClass()
    {
        TestInt = 11;    // CS0198, constructor is not static and readonly field is
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0199

错误消息

无法对静态只读字段传递 `ref` 或 `out` (静态构造函数中除外)

在构造函数中作为 `ref` 或 `out` 参数传递的 `readonly` 变量，必须具有与构造函数相同的 `static` 用法。有关更多信息，请参见[传递参数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0199：

```
// CS0199.cs
class MyClass
{
    public static readonly int TestInt = 6;

    static void TestMethod(ref int testInt)
    {
        testInt = 0;
    }

    MyClass()
    {
        TestMethod(ref TestInt);      // CS0199, TestInt is static
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0200

错误消息

无法对属性或索引器“property”赋值 — 它是只读的

试图对属性赋值，但该属性没有 set 访问器。通过添加 set 访问器解决该错误。有关更多信息，请参见[如何：声明和使用读/写属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0200：

```
// CS0200.cs
public class MainClass
{
    // private int _mi;
    int I
    {
        get
        {
            return 1;
        }

        // uncomment the set accessor and declaration for _mi
        /*
        set
        {
            _mi = value;
        }
        */
    }

    public static void Main ()
    {
        MainClass II = new MainClass();
        II.I = 9;    // CS0200
    }
}
```

编译器错误 CS0201

错误消息

只有 assignment、call、increment、decrement 和 new 对象表达式可用作语句

编译器在遇到无意义的语句时会生成错误。

示例

下面的示例生成 CS0201。

```
// CS0201.cs
public class MainClass
{
    public static void Main()
    {
        2 * 3;    // CS0201
    }
}
```

下面的示例生成 CS0201。

```
// CS0201_b.cs
// compile with: /target:library
public class MyList<T>
{
    public void Add(T x)
    {
        int i = 0;
        if ( (object)x == null)
        {
            checked(i++);    // CS0201

            // OK
            checked {
                i++;
            }
        }
    }
}
```

编译器错误 CS0202

错误消息

foreach 要求“type.GetEnumerator()”的返回类型“type”必须有合适的公共 MoveNext 方法和公共 Current 属性

用于启用使用 foreach 语句的函数 [GetEnumerator](#) 无法返回指针或数组; 它必须返回能够充当枚举数的类的实例。充当枚举数的适当要求包括一个公共 Current 属性和一个公共 MoveNext 方法。

注意

在 C# 2.0 中, 编译器将自动为您生成 Current 和 MoveNext。有关更多信息, 请参见[泛型接口 \(C# 编程指南\)](#)中的代码示例。

下面的示例生成 CS0202:

```
// CS0202.cs

public class C1
{
    public int Current
    {
        get
        {
            return 0;
        }
    }

    public bool MoveNext ()
    {
        return false;
    }

    public static implicit operator C1 (int c1)
    {
        return 0;
    }
}

public class C2
{
    public int Current
    {
        get
        {
            return 0;
        }
    }

    public bool MoveNext ()
    {
        return false;
    }

    public C1[] GetEnumerator ()
    // try the following line instead
    // public C1 GetEnumerator ()
    {
        return null;
    }
}

public class MainClass
{
    public static void Main ()
    {
```

```
C2 c2 = new C2();

foreach (C1 x in c2)    // CS0202
{
    System.Console.WriteLine(x.Current);
}

}
```

编译器错误 CS0204

错误消息

只允许 65535 个局部变量

方法中已经超过了 65535 个局部变量的限制。

编译器错误 CS0205

错误消息

无法调用抽象基成员：“method”

无法调用 `abstract` 方法，因为它没有方法体。有关更多信息，请参见[抽象类、密封类及类成员 \(C# 编程指南\)](#)。

下面的示例生成 CS0205：

```
// CS0205.cs
abstract public class MyClass
{
    abstract public void mf();
}

public class MyClass2 : MyClass
{
    public override void mf()
    {
        base.mf(); // CS0205, delete this line
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0206

错误消息

属性或索引器不得作为 out 或 ref 参数传递

属性不可作为 ref 或 out 参数传递。有关更多信息, 请参见[传递参数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0206:

```
// CS0206.cs
public class MyClass
{
    public static int P
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }

    public static void MyMeth(ref int i)
    // public static void MyMeth(int i)
    {
    }

    public static void Main()
    {
        MyMeth(ref P);    // CS0206
        // try the following line instead
        // MyMeth(P);    // CS0206
    }
}
```

编译器错误 CS0208

错误消息

无法获取托管类型("type")的地址或大小或声明指向它的指针

即使与 [unsafe](#) 关键字一起使用时，也不允许获取托管对象的地址或大小或者声明指向托管类型的指针。有关更多信息，请参见[不安全代码和指针 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0208：

```
// CS0208.cs
// compile with: /unsafe

class S
{
    public int a = 98;
}

public class MyClass
{
    unsafe public static int Main()
    {
        S s = new S(); // S is managed
        S * s2 = &s;   // CS0208
        return 1;
    }
}
```

编译器错误 CS0209

错误消息

在固定语句中声明的局部变量类型必须是指针类型

在 [fixed 语句](#) 中声明的变量必须是指针。有关更多信息, 请参见[不安全代码和指针 \(C# 编程指南\)](#)。

下面的示例生成 CS0209:

```
// CS0209.cs
// compile with: /unsafe

class Point
{
    public int x, y;
}

public class MyClass
{
    unsafe public static void Main()
    {
        Point pt = new Point();

        fixed (int i)      // CS0209
        {
        }
        // try the following lines instead
        /*
        fixed (int* p = &pt.x)
        {
        }
        fixed (int* q = &pt.y)
        {
        }
        */
    }
}
```

编译器错误 CS0210

错误消息

必须在 fixed 或者 using 语句声明中提供初始值设定项

必须在 [fixed 语句](#) 中声明并初始化变量。有关更多信息, 请参见[不安全代码和指针 \(C# 编程指南\)](#)。

下面的示例生成 CS0210:

```
// CS0210a.cs
// compile with: /unsafe

class Point
{
    public int x, y;
}

public class MyClass
{
    unsafe public static void Main()
    {
        Point pt = new Point();

        fixed (int i)    // CS0210
        {
        }
        // try the following lines instead
        /*
        fixed (int* p = &pt.x)
        {
        }
        fixed (int* q = &pt.y)
        {
        }
        */
    }
}
```

下面的示例也生成 CS0210, 因为 [using 语句](#) 没有初始值设定项。

```
// CS0210b.cs

using System.IO;
class Test
{
    static void Main()
    {
        using (StreamWriter w) // CS0210
        // Try this line instead:
        // using (StreamWriter w = new StreamWriter("TestFile.txt"))
        {
            w.WriteLine("Hello there");
        }
    }
}
```

编译器错误 CS0211

错误消息

无法获取给定表达式的地址

可以获取字段、局部变量和指针间接寻址的地址，但不能获取诸如两个局部变量之和的地址。有关更多信息，请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例生成 CS0211：

```
// CS0211.cs
// compile with: /unsafe

public class MyClass
{
    unsafe public void mf()
    {
        int a = 0, b = 0;
        int *i = &(a + b);    // CS0211, the addition of two local variables
        // try the following line instead
        // int *i = &a;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0212

错误消息

只能获取 fixed 语句初始值设定项内的未固定表达式的地址

有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例显示如何获取非固定表达式的地址。下面的示例生成 CS0212。

```
// CS0212a.cs
// compile with: /unsafe /target:library

public class A {
    public int iField = 5;

    unsafe public void mf() {
        A a = new A();
        int* ptr = &a.iField;    // CS0212
    }

    // OK
    unsafe public void mf2() {
        A a = new A();
        fixed (int* ptr = &a.iField) {}
    }
}
```

下面的示例也生成 CS0212 并显示如何纠正该错误:

```
// CS0212b.cs
// compile with: /unsafe /target:library
using System;

public class MyClass
{
    unsafe public void mf()
    {
        // Null-terminated ASCII characters in an sbyte array
        sbyte[] sbArr1 = new sbyte[] { 0x41, 0x42, 0x43, 0x00 };
        sbyte* pAsciiUpper = &sbArr1[0];    // CS0212
        // To resolve this error, delete the previous line and
        // uncomment the following code:
        // fixed (sbyte* pAsciiUpper = sbArr1)
        // {
        //     String szAsciiUpper = new String(pAsciiUpper);
        // }
    }
}
```

编译器错误 CS0213

错误消息

不能使用固定语句来获取已固定的表达式的地址

`unsafe` 方法中的局部变量或某个参数已固定(在堆栈上), 因此无法在 `fixed` 表达式中获取这两个变量的任何一个的地址。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0213。

```
// CS0213.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int i = 45;
        fixed (int *j = &i) { } // CS0213
        // try the following line instead
        // int* j = &i;

        int[] a = new int[] {1,2,3};
        fixed (int *b = a)
        {
            fixed (int *c = b) { } // CS0213
            // try the following line instead
            // int *c = b;
        }
    }
}
```

编译器错误 CS0214

错误消息

指针和固定大小缓冲区只能在不安全的上下文中使用

指针只能与 `unsafe` 关键字一起使用。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例生成 CS0214:

```
// CS0214.cs
// compile with: /target:library /unsafe
public struct S
{
    public int a;
}

public class MyClass
{
    public static void Test()
    {
        S s = new S();
        S * s2 = &s;      // CS0214
        s2->a = 3;       // CS0214
        s.a = 0;
    }

    // OK
    unsafe public static void Test2()
    {
        S s = new S();
        S * s2 = &s;
        s2->a = 3;
        s.a = 0;
    }
}
```

编译器错误 CS0215

错误消息

运算符 True 或 False 的返回类型必须是 bool

用户定义的 `true` 和 `false` 运算符的返回类型必须是 `bool`。有关更多信息, 请参见[可重载运算符 \(C# 编程指南\)](#)。

下面的示例生成 CS0215:

```
// CS0215.cs
class MyClass
{
    public static int operator true (MyClass MyInt)    // CS0215
    // try the following line instead
    // public static bool operator true (MyClass MyInt)
    {
        return true;
    }

    public static int operator false (MyClass MyInt)   // CS0215
    // try the following line instead
    // public static bool operator false (MyClass MyInt)
    {
        return true;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0216

错误消息

运算符“operator”还要求定义匹配的运算符“missing_operator”

用户定义的 `true` 运算符需要用户定义的 `false` 运算符，反之亦然。有关更多信息，请参见[运算符 \(C# 编程指南\)](#)。

下面的示例生成 CS0216：

```
// CS0216.cs
class MyClass
{
    public static bool operator true (MyClass MyInt)    // CS0216
    {
        return true;
    }

    // to resolve, uncomment the following operator definition
/*
    public static bool operator false (MyClass MyInt)
    {
        return true;
    }
*/
}

public static void Main()
{
}
```

编译器错误 CS0217

错误消息

为了可以像短路运算符一样应用, 用户定义的逻辑运算符("operator")必须具有与它的两个参数的类型相同的返回类型。

如果将运算符定义为用户定义类型, 然后尝试将运算符用作短路运算符, 则用户定义的运算符必须具有相同类型的参数和返回值。有关短路运算符的更多信息, 请参见 [&& 运算符](#) 和 [|| 运算符](#)。

下面的示例生成 CS0217:

```
// CS0217.cs
using System;

public class MyClass
{
    public static bool operator true (MyClass f)
    {
        return false;
    }

    public static bool operator false (MyClass f)
    {
        return false;
    }

    public static implicit operator int(MyClass x)
    {
        return 0;
    }

    public static int operator & (MyClass f1, MyClass f2)    // CS0217
    // try the following line instead
    // public static MyClass operator & (MyClass f1, MyClass f2)
    {
        return new MyClass();
    }

    public static void Main()
    {
        MyClass f = new MyClass();
        int i = f && f;
    }
}
```

请参见

参考

[可重载运算符\(C# 编程指南\)](#)

编译器错误 CS0218

错误消息

类型("type")必须包含运算符 true 和运算符 false 的声明

如果将运算符定义为用户定义的类型，然后尝试将运算符用作短路运算符，则用户定义的运算符必须定义了 true 运算符和 false 运算符。有关短路运算符的更多信息，请参见 [&& 运算符](#) 和 [|| 运算符](#)。

下面的示例生成 CS0218：

```
// CS0218.cs
using System;
public class MyClass
{
    // uncomment these operator declarations to resolve this CS0218
    /*
    public static bool operator true (MyClass f)
    {
        return false;
    }

    public static bool operator false (MyClass f)
    {
        return false;
    }
    */

    public static implicit operator int(MyClass x)
    {
        return 0;
    }

    public static MyClass operator & (MyClass f1, MyClass f2)
    {
        return new MyClass();
    }

    public static void Main()
    {
        MyClass f = new MyClass();
        int i = f && f;    // CS0218, requires operators true and false
    }
}
```

请参见

概念

[转换运算符 \(C# 编程指南\)](#)

编译器错误 CS0220

错误消息

在检查模式下，运算在编译时溢出

[checked](#)(这是默认设置)检测到某个运算并导致了数据丢失。为解决该错误，请更正赋值输入或使用 [unchecked](#)。有关更多信息，请参见 [Checked 和 Unchecked\(C# 参考\)](#)。

下面的示例生成 CS0220:

```
// CS0220.cs
using System;

class TestClass
{
    const int x = 1000000;
    const int y = 1000000;

    public int MethodCh()
    {
        int z = (x * y);    // CS0220
        return z;
    }

    public int MethodUnCh()
    {
        unchecked
        {
            int z = (x * y);
            return z;
        }
    }

    public static void Main()
    {
        TestClass myObject = new TestClass();
        Console.WriteLine("Checked : {0}", myObject.MethodCh());
        Console.WriteLine("Unchecked: {0}", myObject.MethodUnCh());
    }
}
```

编译器错误 CS0221

错误消息

常数值“value”无法转换为“type”(使用“unchecked”语法重写)

[checked](#)(默认情况下是打开的)检测到某个赋值运算会导致数据丢失。为解决该错误,请更正赋值或使用[unchecked](#)。有关更多信息,请参见[Checked 和 Unchecked\(C# 参考\)](#)。

下面的示例生成 CS0221:

```
// CS0221.cs
public class MyClass
{
    public static void Main()
    {
        // unchecked
        // {
        int a = (int)0xFFFFFFFF;    // CS0221
        a++;
        // }
    }
}
```

编译器错误 CS0225

错误消息

params 参数必须是一维数组

当使用 `params` 关键字时，必须指定该数据类型的一维数组。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0225：

```
// CS0225.cs
public class MyClass
{
    public static void TestParams(params int a)    // CS0225
    // try the following line instead
    // public static void TestParams(params int[] a)
    {
    }

    public static void Main()
    {
        TestParams(1);
    }
}
```

编译器错误 CS0227

错误消息

不安全代码只会在使用 /unsafe 编译的情况下出现

如果源代码包含 `unsafe` 关键字, 还必须使用 `/unsafe` 编译器选项。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例(在不使用 `/unsafe` 编译时)将生成 CS0227:

```
// CS0227.cs
public class MyClass
{
    unsafe public static void Main()    // CS0227
    {
    }
}
```

编译器错误 CS0228

错误消息

"type"不包含"member"的定义, 或者它是不可访问的

可能已经替换了 **System.Hashtable**、**System.String** 或 **System.Array** 的系统版本, 并且该替换不包含 *member*。

否则, 您可能需要修复或重新安装 Visual Studio。

编译器错误 CS0229

错误消息

"member1" 和 "member2" 之间具有二义性

不同接口的成员具有相同的名称。如果要保留相同的名称，则必须限定这些名称。有关更多信息，请参见[接口 \(C# 编程指南\)](#)。

注意

在某些情况下，此二义性可以通过使用 [using](#) 别名向标识符提供明确的前缀来解决。

示例

下面的示例生成 CS0229：

```
// CS0229.cs

interface IList
{
    int Count
    {
        get;
        set;
    }

    void Counter();
}

interface Icounter
{
    double Count
    {
        get;
        set;
    }
}

interface IListCounter : IList , Icounter {}

class MyClass
{
    void Test(IListCounter x)
    {
        x.Count = 1; // CS0229
        // Try one of the following lines instead:
        // ((IList)x).Count = 1;
        // or
        // ((Icounter)x).Count = 1;
    }

    public static void Main() {}
}
```

编译器错误 CS0230

错误消息

在 foreach 语句中，类型和标识符都是必需的

foreach 语句格式不对。

下面的示例生成 CS0230：

```
// CS0230.cs
using System;

class MyClass
{
    public static void Main()
    {
        int[] myarray = new int[3] {1,2,3};

        foreach (int in myarray) // CS0230
        // try the following line instead
        // foreach (int x in myarray)
        {
            Console.WriteLine(x);
        }
    }
}
```

编译器错误 CS0231

错误消息

params 参数必须是形参表中的最后一个参数。

params 参数支持可变的参数数目，并且必须位于所有其他参数之后。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0231：

```
// CS0231.cs
class Test
{
    public void TestMethod(params int[] p, int i) {} // CS0231
    // To resolve the error, use the following line instead:
    // public void TestMethod(int i, params int[] p) {}

    static void Main()
    {
    }
}
```

编译器错误 CS0233

错误消息

"identifier"没有预定义大小, 因此 sizeof 只能用于不安全的上下文中(请考虑使用 System.Runtime.InteropServices.Marshal.SizeOf)

sizeof 运算符只能用于为编译时常数的类型。如果您遇到此错误, 请确保在编译时可以确定标识符的大小。如果不能确定, 则用 SizeOf 替代 sizeof。

示例

下面的示例生成 CS0233:

```
// CS0233.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Sequential)]
public struct S
{
    public int a;
}

public class MyClass
{
    public static void Main()
    {
        S myS = new S();
        Console.WriteLine(sizeof(S));    // CS0233
        // Try the following line instead:
        // Console.WriteLine(Marshal.SizeOf(myS));
    }
}
```

编译器错误 CS0234

错误消息

命名空间“namespace”中不存在类型或命名空间名称“name”(是否缺少程序集引用?)

应为某个类型。可能的原因包括:

- 在编译中未引用包含某个类型定义的程序集。请使用 [/reference\(导入元数据\)](#) 指定程序集。
- 将某个变量名传递给了 `typeof` 运算符。

有关如何在开发环境中添加引用的信息, 请参见[“添加引用”对话框](#)。

下面的示例生成 CS0234:

```
// CS0234.cs
public class C
{
    public static void Main()
    {
        System.DateTime x = new System.DateTim();    // CS0234
        // try the following line instead
        // System.DateTime x = new System.DateTime();
    }
}
```

编译器错误 CS0236

错误消息

字段初始值设定项不能引用非静态字段、方法或属性“field”

实例字段不能用于初始化方法之外的其他实例字段。如果您正尝试在方法之外初始化一个变量，请考虑在类构造函数内部执行初始化。有关更多信息，请参见[方法\(C# 编程指南\)](#)。

下面的示例生成 CS0236:

```
// CS0236.cs
public class MyClass
{
    public int i = 5;
    public int j = i; // CS0236
    public int k;      // initialize in constructor

    MyClass()
    {
        k = i;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0238

错误消息

因为“member”不是重写，所以无法将其密封

在还未用 `override` 进行标记的成员上使用了 `sealed`。有关更多信息，请参见[继承\(C# 编程指南\)](#)。

下面的示例生成 CS0238：

```
// CS0238.cs
abstract class MyClass
{
    public abstract void f();
}

class MyClass2 : MyClass
{
    public static void Main()
    {

        public sealed void f() // CS0238
        // Try the following definition instead:
        // public override sealed void f()
        {
        }
    }
}
```

编译器错误 CS0239

错误消息

"member": 无法重写继承成员 "inherited member", 因为它已被密封

成员不能 `overridesealed` 继承成员。有关更多信息, 请参见 [Checked 和 Unchecked\(C# 参考\)](#)。

下面的示例生成 CS0239:

```
// CS0239.cs
abstract class MyClass
{
    public abstract void f();
}

class MyClass2 : MyClass
{
    public static void Main()
    {

        public override sealed void f()
        {
        }
    }
}

class MyClass3 : MyClass2
{
    public override void f()    // CS0239
    // Try the following definition instead:
    // public new void f()
    {
    }
}
```

编译器错误 CS0241

错误消息

不允许有默认参数说明符

方法参数不能有默认值。如果要获得同样的效果，请使用方法重载。有关更多信息，请参见[传递参数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0241。此外，该示例还演示如何使用重载模拟具有默认参数的方法。

```
// CS0241.cs
public class A
{
    public void Test(int i = 9) {}    // CS0241
}

public class B
{
    public void Test() { Test(9); }
    public void Test(int i) {}
}

public class C
{
    public static void Main()
    {
        B x = new B();
        x.Test();
    }
}
```

编译器错误 CS0242

错误消息

相关操作在 void 指针上未定义

不允许增加 void 指针。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例生成 CS0242:

```
// CS0242.cs
// compile with: /unsafe
class TestClass
{
    public unsafe void Test()
    {
        void * p = null;
        p++; // CS0242, incrementing a void pointer not allowed
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0243

错误消息

Conditional 属性在“method”上无效，因为该属性是重写方法

在用 `override` 关键字标记的方法上不允许有 `Conditional` 属性。有关更多信息，请参见[了解何时使用 Override 和 New 关键字 \(C# 编程指南\)](#)。

编译器从不绑定到重写方法；它只绑定到基方法，而且公共语言运行库在适当的时候调用重写。

下面的示例生成 CS0243：

```
// CS0243.cs
// compile with: /target:library
public class MyClass
{
    public virtual void M() {}

public class MyClass2 : MyClass
{
    [System.Diagnostics.ConditionalAttribute("MySymbol")] // CS0243
    // remove Conditional attribute or remove override keyword
    public override void M() {}
}
```

编译器错误 CS0244

错误消息

"is" 和 "as" 在指针类型上都无效

在指针类型上使用 `is` 和 `as` 关键字无效。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例生成 CS0244:

```
// CS0244.cs
// compile with: /unsafe

class UnsafeTest
{
    unsafe static void SquarePtrParam (int* p)
    {
        bool b = p is object;    // CS0244 p is pointer
    }

    unsafe public static void Main()
    {
        int i = 5;
        SquarePtrParam (&i);
    }
}
```

编译器错误 CS0245

错误消息

不能直接调用析构函数和 object.Finalize。请考虑调用 IDisposable.Dispose(如可用)。

有关更多信息, 请参见[垃圾回收编程原理](#)和[析构函数\(C# 编程指南\)](#)。

下面的示例生成 CS0245:

```
// CS0245.cs
using System;
using System.Collections;

class MyClass // : IDisposable
{
    /*
    public void Dispose()
    {
        // cleanup code goes here
    }
    */

    void m()
    {
        this.Finalize();    // CS0245
        // this.Dispose();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0246

错误消息

找不到类型或命名空间名称“type(namespace”(是否缺少 using 指令或程序集引用?)

未找到类型。可能忘了引用 ([/reference](#)) 包含该类型的程序集, 或者可能未用 [using 指令](#) 正确地限定其用法。

出现此错误的几个原因是:

1. 试图使用的类型名称或命名空间名称可能拼写错误(包括正确的大小写)。没有正确的名称, 编译器就无法找到在代码中引用的类型或命名空间的定义。这种情况最常发生, 原因是 C# 区分大小写, 而在引用类型时没有使用正确的大小写。例如, `Dataset ds;` 会生成 CS0246; 请注意, `Dataset` 中的 s 没有大写。
 2. 如果错误的原因在于命名空间名称, 则可能是没有引用 ([/reference](#)) 包含该命名空间的程序集。例如, 代码可能包含 `using Accessibility;`。但是, 如果项目不引用程序集 `Accessibility.dll`, 则会遇到 CS0246。有关如何在开发环境中添加引用的信息, 请参见[“添加引用”对话框](#)。
 3. 如果错误的原因在于类型名称, 则可能是没有使用正确的 [using](#) 指令, 或没有完全限定该类型的名称。考虑下面的代码行:`DataSet ds;`。为了能够使用 `DataSet` 类型, 您需要做以下两件事。第一, 需要一个对包含 `DataSet` 类型定义的程序集的引用。第二, 需要对 `DataSet` 所在的命名空间使用 [using](#) 指令。例如, 由于 `DataSet` 位于 **System.Data** 命名空间中, 所以需要在代码的开头有以下语句:`using System.Data;`。
- 第二步不是必需的。但如果省略这一步, 则要求在引用 `DataSet` 类型时对它进行完全限定。完全限定 `DataSet` 类型意味着每次在代码中引用它时都要使用命名空间和类型。所以, 如果决定跳过第二步, 您需要将上述声明代码更改
为:`System.Data.DataSet ds;`。
4. 如果是非类型的错误, 说明您可能在需要类型时使用了变量或其他对象。例如, 在 `is` 语句中, 如果您使用 `Type` 对象而不是实际的类型, 将会遇到此错误。

下面的示例生成 CS0246:

```
// CS0246.cs
// using System.Diagnostics;

public class MyClass
{
    [Conditional("A")] // CS0246, uncomment using directive to resolve
    public void Test()
    {
    }

    public static void Main()
    {
    }
}
```

下面是一个示例, 其中使用了 `Type` 类型的对象, 但应该使用实际的类型(上面的第 4 种情况):

```
// CS0246b.cs
using System;

class C
{
    public bool supports(object o, Type t)
    {
        if (o is t) // CS0246 - t is not a type
        {
            return true;
        }
        return false;
    }

    public static void Main()
```

{
}
}

编译器错误 CS0247

错误消息

无法对 stackalloc 采用负值大小

将负数传递给了 `stackalloc` 语句。

下面的示例生成 CS0247:

```
// CS0247.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int *p = stackalloc int [-30]; // CS0247
    }
}
```

编译器错误 CS0248

错误消息

无法创建大小为负值的数组

用负数指定了数组大小。有关更多信息, 请参见[数组 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0248:

```
// CS0248.cs
class MyClass
{
    public static void Main()
    {
        int[] myArray = new int[-3] {1,2,3}; // CS0248, pass a nonnegative number
    }
}
```

编译器错误 CS0249

错误消息

不要重写 `object.Finalize`。相反, 请提供析构函数。

使用析构函数语法指定销毁对象时执行的指令。

有关更多信息, 请参见 [C# 和 C++ 中的析构函数语法](#)。

下面的示例生成 CS0249:

```
// CS0249.cs
class MyClass
{
    protected override void Finalize() // CS0249
    // try the following line instead
    // ~MyClass()
}

public static void Main()
{
}
}
```

编译器错误 CS0250

错误消息

不要直接调用基类 Finalize 方法。它将从析构函数中自动调用。

程序不能试图强制清理基类资源。

有关更多信息, 请参见 [Finalize 方法和析构函数](#)。

下面的示例生成 CS0250:

```
// CS0250.cs
class B
{
}

class C : B
{
    ~C()
    {
        base.Finalize();    // CS0250
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0254

错误消息

固定语句赋值的右边不能是强制转换表达式

`fixed` 表达式的右边不可使用转换。有关更多信息, 请参见[不安全代码和指针\(C# 编程指南\)](#)。

下面的示例生成 CS0254:

```
// CS0254.cs
// compile with: /unsafe
class Point
{
    public uint x, y;
}

class FixedTest
{
    unsafe static void SquarePtrParam (int* p)
    {
        *p *= *p;
    }

    unsafe public static void Main()
    {
        Point pt = new Point();
        pt.x = 5;
        pt.y = 6;

        fixed (int* p = (int*)&pt.x)    // CS0254
        // try the following line instead
        // fixed (uint* p = &pt.x)
        {
            SquarePtrParam ((int*)p);
        }
    }
}
```

编译器错误 CS0255

错误消息

stackalloc 不能用在 catch 或 finally 块中

stackalloc 关键字不可用在 catch 或 finally 块中。有关更多信息, 请参见[异常和异常处理\(C# 编程指南\)](#)。

下面的示例生成 CS0255:

```
// CS0255.cs
// compile with: /unsafe
using System;

public class TestTryFinally
{
    public static unsafe void Test()
    {
        int i = 123;
        string s = "Some string";
        object o = s;

        try
        {
            // Conversion is not valid; o contains a string not an int
            i = (int) o;
        }

        finally
        {
            Console.WriteLine("i = {0}", i);
            int* fib = stackalloc int[100];    // CS0255
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0260

错误消息

类型“type”的声明上缺少分部修饰符; 存在此类型的其他分部声明

此错误指示找到了多个具有相同名称的类声明，并且这些声明中至少有一个但不是全部被声明为分部。如果您打算在若干部分中定义一个类，则所有那些部分必须用关键字 **partial** 声明。如果您创建的新类碰巧与同一命名空间中的其他位置声明的分部类具有相同的名称，则也会发生此错误。

下面的示例生成 CS0260：

```
// CS0260.cs
class C // CS0260
{
}

partial class C
{
}
```

编译器错误 CS0261

错误消息

"type"的分部声明必须全部是类、构造或接口

如果分部类型在不同的位置声明为不同的构造类型，则会发生此错误。有关更多信息，请参见[分部类定义 \(C# 编程指南\)](#)。

下面的示例生成 CS0261：

```
// CS0261.cs
partial class A // CS0261 - A declared as a class here, but as a struct below
{
}

partial struct A
{
}
```

编译器错误 CS0262

错误消息

"type"的分部声明包含冲突的可访问性修饰符

如果分部类型有不一致的修饰符, 如 public、private、protected、internal 或 abstract, 则将发生此错误。这些修饰符在该类型的
所有分部声明中必须一致。有关更多信息, 请参见[分部类定义\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0262:

```
// CS0262.cs
class A
{
    public partial class C // CS0262
    {
    }
    private partial class C
    {
    }
}
```

编译器错误 CS0263

错误消息

"type"的分部声明不得指定不同的基类

在分部声明中定义类型时, 请在每个分部声明中指定相同的基类型。有关更多信息, 请参见[分部类定义\(C# 编程指南\)](#)。

下面的示例生成 CS0263:

```
// CS0263.cs
// compile with: /target:library
class B1
{
}

class B2
{
}
partial class C : B1 // CS0263 - is the base class B1 or B2?
{
}

partial class C : B2
{
}
```

编译器错误 CS0264

错误消息

"type"的分部声明必须具有相同顺序的相同类型参数名

如果您要在分部声明中定义一个泛型类型，但类型参数的名称或顺序在所有分部声明中不一致，就会发生此错误。若要消除此错误，请检查每个分部声明的类型参数，确保使用的参数名称和顺序相同。有关更多信息，请参见[分部类定义\(C# 编程指南\)](#)和[泛型类型参数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0264。

```
// CS0264.cs

partial class MyClass<T> // CS0264
{
}

partial class MyClass <MyType>
{
}
```

编译器错误 CS0265

错误消息

"type"的分部声明具有与类型参数" type parameter"不一致的约束

当您将某个泛型类定义为分部类，以便其分部定义出现在多个位置，如果该泛型类的约束在两个或更多的位置中不一致或不同，则会发生此错误。如果您在多个位置指定约束，这些约束必须相同。最简单的解决方法是在一个位置指定约束，并在所有其他位置省略它们。有关更多信息，请参见[分部类定义\(C# 编程指南\)](#)和[类型参数的约束\(C# 编程指南\)](#)。

下面的代码生成错误 CS0265。

示例

在此代码中，分部类定义全部位于单个文件中，不过它们也可以分布在多个文件中。

```
// CS0265.cs
public class GenericsErrors
{
    interface IFace1 { }
    interface IFace2 { }
    partial class PartialBadBounds<T> where T : IFace1 { } // CS0265
    partial class PartialBadBounds<T> where T : IFace2 { }
}
```

编译器错误 CS0266

错误消息

无法将类型“type1”隐式转换为“type2”。存在显式转换(是否缺少强制转换?)

如果您的代码试图转换两种不能隐式转换的类型(例如将基类分配给一个缺少显式强制转换的派生类), 将发生此错误。有关更多信息, 请参见[转换运算符\(C# 编程指南\)](#)。

下面的示例生成 CS0266:

```
// CS0266.cs
class MyClass
{
    public static void Main()
    {
        object obj = "MyString";
        // Cannot implicitly convert 'object' to 'MyClass'
        MyClass myClass = obj; // CS0266
        // Try this line instead
        // MyClass c = ( MyClass )obj;
    }
}
```

编译器错误 CS0267

错误消息

分部修饰符只能仅挨在“class”、“struct”或“interface”之前出现

在类、结构或接口的声明中，**partial** 修饰符的位置不正确。若要修复此错误，请重排修饰符的顺序。有关更多信息，请参见[分部类定义\(C# 编程指南\)](#)。

下面的示例生成 CS0267：

```
// CS0267.cs
public partial class MyClass
{
    public MyClass()
    {
    }
}

partial public class MyClass // CS0267
// Try this line instead:
// public partial class MyClass
{
    public void Foo()
    {

    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0268

错误消息

导入的类型“type”无效。它包含一个循环基类依赖项。

如果从另一种语言导入的类型有循环基类依赖项，则发生此错误。此类型不能在 C# 程序中使用。要解决此错误，请在所有引用的程序集或模块中检查从其他语言导入的类型。

编译器错误 CS0269

错误消息

使用了未赋值的 out 参数“parameter”

编译器无法验证在使用 out 参数之前是否已给它赋值。它的值在赋值时可能未定义。在访问值之前，请确保初始化 **out** 参数。如果您需要使用传入的变量的值，请改用 **ref** 参数。有关更多信息，请参见[传递参数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0269：

```
// CS0269.cs
class C
{
    public static void F(out int i)
    // Try this instead:
    // public static void F(ref int i)
    {
        int k = i; // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

如果变量的初始化发生在 try 块中，则也会出现此错误，因为编译器无法验证 try 块是否可以成功执行：

```
// CS0269b.cs
class C
{
    public static void F(out int i)
    {
        try
        {
            // Assignment occurs, but compiler can't verify it
            i = 1;
        }
        catch
        {
        }

        int k = i; // CS0269
        i = 1;
    }

    public static void Main()
    {
        int myInt;
        F(out myInt);
    }
}
```

编译器错误 CS0270

错误消息

不能在变量声明中指定数组大小(请尝试使用“new”表达式初始化)

将大小指定为数组声明的一部分时会发生此错误。若要解决此错误, 请使用 [new 运算符](#)表达式。

下面的示例生成 CS0270:

```
// CS0270.cs
// compile with: /t:module

public class Test
{
    int[10] a;    // CS0270
    // To resolve, use the following line instead:
    // int[] a = new int[10];
}
```

编译器错误 CS0271

错误消息

由于 get 访问器不可访问，因此不能在此上下文中使用属性或索引器“property/indexer”

当您试图访问一个不可访问的 **get** 访问器时，会发生此错误。若要解决此错误，请提高访问器的可访问性，或者更改调用位置。有关更多信息，请参见[访问器可访问性和属性\(C# 编程指南\)](#)。

下面的示例生成 CS0271：

```
// CS0271.cs
public class MyClass
{
    public int Property
    {
        private get { return 0; }
        set { }
    }

    public int Property2
    {
        get { return 0; }
        set { }
    }
}

public class Test
{
    public static void Main(string[] args)
    {
        MyClass c = new MyClass();
        int a = c.Property;    // CS0271
        int b = c.Property2;   // OK
    }
}
```

编译器错误 CS0272

错误消息

由于无法访问 set 访问器, 属性或索引器“property/indexer”不能在此上下文中使用

当 **set** 访问器不能由程序代码访问时, 将发生此错误。若要解决此错误, 请提高访问器的可访问性, 或者更改调用位置。有关更多信息, 请参见[非对称访问器可访问性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0272:

```
// CS0272.cs
public class MyClass
{
    public int Property
    {
        get { return 0; }
        private set { }
    }
}

public class Test
{
    static void Main()
    {
        MyClass c = new MyClass();
        c.Property = 10;      // CS0272
        // To resolve, remove the previous line
        // or use an appropriate modifier on the set accessor.
    }
}
```

编译器错误 CS0273

错误消息

"property_accessor"访问器的可访问性修饰符必须比属性或索引器"property"具有更强的限制

set/get 访问器的可访问性修饰符必须比属性或索引器"property/indexer"具有更强的限制

如果您用一个访问修饰符声明一个属性或索引器，而该访问修饰符的限制性低于该属性或索引器的其中一个访问器的访问修饰符，则会发生此错误。若要解决此错误，请对属性或 set 访问器使用适当的访问修饰符。有关更多信息，请参见[访问器可访问性](#)。

示例

此示例包含一个具有内部 set 方法的内部属性。下面的示例生成 CS0273。

```
// CS0273.cs
// compile with: /target:library
public class MyClass
{
    internal int Property
    {
        get { return 0; }
        internal set {} // CS0273
        // try the following line instead
        // private set {}
    }
}
```

编译器错误 CS0274

错误消息

不能为属性或索引器“property/indexer”的两个访问器同时指定可访问性修饰符

当您声明一个在两个访问器上都有访问修饰符的属性或索引器时，将出现此错误。若要解决此错误，请仅在两个访问器中的一个上使用访问修饰符。有关更多信息，请参见[访问器可访问性](#)。

下面的示例生成 CS0274：

```
// CS0274.cs
public class MyClass
{
    public int Property // CS0274
    {
        public get { return 0; }
        protected set { }
    }
}
```

编译器错误 CS0275

错误消息

"accessor": 可访问性修饰符不能在接口中的访问器上使用

当您在接口中的属性或索引器的任何一个访问器上使用访问修饰符时，将出现此错误。若要解决此错误，请移除访问修饰符。

示例

下面的示例生成 CS0275：

```
// CS0275.cs
public interface MyInterface
{
    int Property
    {
        get;
        internal set; // CS0275
    }
}
```

编译器错误 CS0276

错误消息

"property/indexer":仅当属性或索引器同时具有 get 访问器和 set 访问器时，才能对访问器使用可访问修饰符

当您仅使用一个访问器声明属性或索引器，并对该访问器使用访问修饰符时，会出现此错误。若要解决此错误，请移除访问修饰符或添加另一个访问器。

示例

下面的示例生成 CS0276：

```
// CS0276.cs
public class MyClass
{
    public int Property
    {
        protected set { } // CS0276
    }
    public int Property2
    {
        internal get { } // CS0276
    }
}
```

编译器错误 CS0277

错误消息

"class"不实现接口成员"accessor"。"class accessor"不是公共的

当您尝试实现某个接口的属性，但在类中实现的属性访问器不是公共的时，则会发生此错误。实现接口成员的方法需要具有公共访问性。若要解决此错误，请移除属性访问器的访问修饰符。

示例

下面的示例生成 CS0277：

```
// CS0277.cs
public interface MyInterface
{
    int Property
    {
        get;
        set;
    }
}

public class MyClass : MyInterface // CS0277
{
    public int Property
    {
        get { return 0; }
        // Try this instead:
        //set { }
        protected set { }
    }
}
```

编译器错误 CS0281

错误消息

已授予对“AssemblyName1”的友元访问，但输出程序集的名称为“AssemblyName2”。请尝试添加对“AssemblyName1”的引用，或者更改输出程序集的名称以使其匹配。

友元访问是一种新的公共语言运行库 (CLR) 功能，它可以使一个程序集看到另一个程序集的非公共类型。当授予友元访问的程序集为被授予程序集指定错误的名称时，将会出现此错误。有关更多信息，请参见[友元程序集 \(C# 编程指南\)](#)。

示例

下面的代码示例序列将生成 CS0281。

按照下面的方式生成用于创建具有强名称的程序集的文件：

- sn -d CS0281.snk
- sn -k CS0281.snk
- sn -i CS0281.snk CS0281.snk
- sn -pc CS0281.snk key.publickey
- sn -tp key.publickey

```
// CS0281.cs
// compile with: /target:library /keyfile:CS0281.snk
public class A {}
```

```
// CS0281_b.cs
// compile with: /target:library /keyfile:CS0281.snk /reference:CS0281.dll
[assembly:System.Runtime.CompilerServices.InternalVisibleTo("CS0281 , PublicKey=0024000004
8000009400000006020000002400005253413100040000010001004b2d4d56af7c50be2fcbbf97cb880b9e73ad8
4467a587191fef63aadcc118a96cecf9d508cd679c907b6e20f71684300bdc2c0a851019af0c96b29bf8f1339753
276041aefd67db46139e6348b3a12f29537b4dc6c2c19829df2c9ed6803f3c63c3b84cfa2728849386aea575c54
3a5f70fa85793d2946f15f7fe1ccb0c5e8fe0")]
class B : A {}
```

下面的示例生成 CS0281。

注意，此示例创建的输出文件与第一个示例创建的输出文件同名。若要解决此问题，请不要更改组件的程序集属性并添加类 C。

```
// CS0281_c.cs
// compile with: /target:library /out:CS0281.dll /keyfile:CS0281.snk /reference:CS0281_b.dll
1
// CS0281 expected
[assembly:System.Reflection.AssemblyVersion("3")]
[assembly:System.Reflection.AssemblyCulture("en-us")]
class C : B {}
public class A {}
```

编译器错误 CS0283

错误消息

不能将类型“type”声明为 const

常量声明中指定的类型必须是 **byte**、**char**、**short**、**int**、**long**、**float**、**double**、**decimal**、**bool**、**string**、枚举类型、或被赋予空值的引用类型。每个常数表达式产生的值必须属于目标类型，或者属于可隐式转换为目标类型的类型。

示例

下面的示例生成 CS0283。

```
// CS0283.cs
struct MyTest
{
}
class MyClass
{
    // To resolve the error but retain the "const-ness",
    // change const to readonly.
    const MyTest test = new MyTest();    // CS0283

    public static int Main() {
        return 1;
    }
}
```

编译器错误 CS0304

错误消息

变量类型“type”没有 new() 约束，因此无法创建该类型的实例

当您使用 new 创建某个类型变量的实例时，如果该类型变量未指定 new() constraint，则会发生此错误。除非使用 new() 约束确保存在默认的构造函数，否则不能使用 new 直接调用未知类型的构造函数。如果无法使用 new 约束，可考虑使用 typeof 表达式访问所需的构造函数。

下面的示例生成 CS0304：

```
// CS0304.cs
// compile with: /target:library
class C<T>
{
    T t = new T(); // CS0304
}
```

类方法中同样不允许这种形式的 new 语句：

```
// CS0304_2.cs
// compile with: /target:library
class C<T>
{
    public void f()
    {
        T t = new T(); // CS0304
    }
}
```

编译器错误 CS0305

错误消息

使用泛型类型“generic type”需要“number”类型参数

当未找到预期的类型参数数目时会发生此错误。若要解决 CS0305，请使用所需数目的类型参数。

示例

下面的示例生成 CS0305。

```
// CS0305.cs
public class MyList<T> {}
public class MyClass<T> {}

class MyClass
{
    public static void Main()
    {
        MyList<MyClass, MyClass> list1 = new MyList<MyClass>();    // CS0305
        MyList<MyClass> list2 = new MyList<MyClass>();    // OK
    }
}
```

编译器错误 CS0306

错误消息

类型“type”不能用作类型参数

不允许将该类型用作类型参数。这可能是因为该类型是一个指针类型。

下面的示例生成 CS0306:

```
// CS0306.cs
class C<T>
{
}

class M
{
    // CS0306 - int* not allowed as a type parameter
    C<int*> f;
}
```

编译器错误 CS0307

错误消息

"construct""identifier"不能与类型参数一起使用

指定的构造不是一个类型或方法，而类型和方法是唯一能接受泛型参数的构造。请移除尖括号中的类型参数。如果需要泛型，请将泛型构造声明为泛型类型或方法。

下面的示例生成 CS0307：

```
// CS0307.cs
class C
{
    public int P { get { return 1; } }
    public static void Main()
    {
        C c = new C();
        int p = c.P<int>(); // CS0307 - C.P is a property
        // Try this instead
        // int p = c.P;
    }
}
```

编译器错误 CS0308

错误消息

非泛型类型或方法“identifier”不能与类型参数一起使用。

方法或类型不是泛型的，但它却与类型参数一起使用。若要避免此错误，请移除尖括号和类型参数，或者将方法或类型重新声明为泛型类型或方法。

下面的示例生成 CS0308：

```
// CS0308a.cs
class MyClass
{
    public void F() {}
    public static void Main()
    {
        F<int>(); // CS0308 - F is not generic.
        // Try this instead:
        // F();
    }
}
```

下面的示例生成 CS0308。若要解决此错误，请使用指令“using System.Collections.Generic”。

```
// CS0308b.cs
// compile with: /t:library
using System.Collections;
// To resolve, uncomment the following line:
// using System.Collections.Generic;
public class MyStack<T>
{
    // Store the elements of the stack:
    private T[] items = new T[100];
    private int stack_counter = 0;

    // Define the iterator block:
    public IEnumerator<T> GetEnumerator() // CS0308
    {
        for (int i = stack_counter - 1 ; i >= 0; i--)
            yield return items[i];
    }
}
```

编译器错误 CS0309

错误消息

类型“typename”必须可以转换为“constraint type”才能用作泛型类型或方法“generic”中的参数“parameter”。

使用泛型类或方法时，必须遵守使用 `where` 关键字对泛型类型施加的约束。违反约束将导致此错误。要修复此错误，应将另一类型传递到泛型类或方法中，或应更改这些约束。

由于 B 不实现 I，而 C<T> 在其约束中指定 T 必须实现 I，以下示例生成 CS0309：

```
// CS0309.cs
using System;

interface I
{
}

class C<T> where T : I
{
}

class B
{
}

class CMain
{
    public static void Main()
    {
        Console.WriteLine(new C<B>()); // CS0309
    }
}
```

编译器错误 CS0310

错误消息

类型“typename”必须具有公共的无参数构造函数，才能用作泛型类型或方法“generic”中的参数“parameter”

泛型类型或方法在它的 where 子句中定义了一个新约束，要求任何类型均必须具有公共的无参数构造函数才能用作该泛型类型或方法的类型参数。若要避免此错误，请确保类型具有正确的构造函数，或者修改泛型类型或方法的约束子句。

示例

下面的示例生成 CS0310：

```
// CS0310.cs
using System;

class G<T> where T : new()
{
    T t;

    public G()
    {
        t = new T();
        Console.WriteLine(t);
    }
}

class B
{
    private B() { }
    // Try this instead:
    // public B() { }
}

class CMain
{
    public static void Main()
    {
        G<B> g = new G<B>();    // CS0310
        Console.WriteLine(g.ToString());
    }
}
```

编译器错误 CS0400

错误消息

未能在全局命名空间中找到类型或命名空间名称“identifier”(是否缺少程序集引用?)

在全局命名空间中未找到由全局范围运算符 (::) 所定范围的标识符。您可能缺少包含该标识符的程序集引用，或者该标识符可能在全局命名空间以外的类或命名空间中声明。如果全局范围标识符没有被声明或拼写错误，可能也会发生此错误。

若要避免此错误，请找到标识符的声明，验证其拼写是否正确，并且如果该声明是在单独的程序集中，请确保您具有适当的程序集引用。如果标识符在另一个类型或命名空间中声明，请在 :: 之后使用完全限定名。下面的示例生成 CS0400：

```
// CS0400.cs
class C
{
    public static void Main()
    {
        // CS0400 - D could not be found
        // in the global namespace.
        global::D d = new global::D();
    }
}
```

编译器错误 CS0401

错误消息

new() 约束必须是指定的最后一个约束

使用多个约束时, 请将所有其他约束列在 new() 约束之前。

示例

下面的示例生成 CS0401。

```
// CS0401.cs
// compile with: /target:library
using System;
class C<T> where T : new(), IDisposable {} // CS0401

class D<T> where T : IDisposable
{
    static void F<U>() where U : new(), IDisposable{} // CS0401
}
```

编译器错误 CS0403

错误消息

无法将 null 转换为类型参数“type parameter”，因为它可能是值类型。请考虑改用 default('T')。

给指定的未知类型的赋值不能为空，因为它可能是一个不允许空赋值的值类型。如果您的泛型类不用来接受值类型，请使用类型约束。如果它可以接受值类型（例如内置类型），您也许能够用表达式 default(T) 替换空赋值，如下面的示例所示。

示例

下面的示例生成 CS0403。

```
// CS0403.cs
// compile with: /target:library
class C<T>
{
    public void f()
    {
        T t = null; // CS0403
        T t2 = default(T); // OK
    }
}

class D<T> where T : class
{
    public void f()
    {
        T t = null; // OK
    }
}
```

编译器错误 CS0404

错误消息

不应为“<”:属性不能为泛型

属性中不允许使用泛型类型参数。移除类型参数和尖括号。

下面的示例生成 CS0404:

```
// CS0404.cs
[MyAttrib<int>] // CS0404
class C
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0405

错误消息

类型参数“type parameter”的重复约束“constraint”

泛型声明的两个约束完全相同。若要消除此错误，请移除重复的约束。

下面的示例生成 CS0405:

```
// CS0405.cs
interface I
{
}

class C<T> where T : I, I // CS0405.cs
{
}
```

编译器错误 CS0406

错误消息

类类型约束“constraint”必须在其他任何约束之前

当泛型类型或方法具有类类型约束时，该约束必须首先列出。若要避免此错误，请将类类型约束移到约束列表的开头。

示例

下面的示例生成 CS0406。

```
// CS0406.cs
// compile with: /target:library
interface I {}
class C {}
class D<T> where T : I, C {}    // CS0406
class D2<T> where T : C, I {}   // OK
```

编译器错误 CS0407

错误消息

"return-type method"具有错误的返回类型

此方法与委托类型不兼容。参数类型匹配，但返回类型不是该委托的正确返回类型。若要避免此错误，请使用其他方法、更改方法的返回类型或更改委托的返回类型。

示例

下面的示例生成 CS0407:

```
// CS0407.cs
public delegate int MyDelegate();

class C
{
    MyDelegate d;

    public C()
    {
        d = new MyDelegate(F); // OK: F returns int
        d = new MyDelegate(G); // CS0407 - G doesn't return int
    }

    public int F()
    {
        return 1;
    }

    public void G()
    {
    }

    public static void Main()
    {
        C c1 = new C();
    }
}
```

编译器错误 CS0409

错误消息

已经为类型参数“type parameter”指定了约束子句。类型参数的所有约束必须在单个 where 子句中指定。

已找到单个类型参数的多个约束子句(where 子句)。移除无关的 where 子句，或对 where 子句进行更正，以便每个子句中有一个唯一的类型参数。

```
// CS0409.cs
interface I
{
}

class C<T1, T2> where T1 : I where T1 : I // CS0409 - T1 used twice
{
}
```

编译器错误 CS0410

错误消息

"method"的重载没有正确的参数类型和返回类型

如果您试图使用参数类型不正确的函数来实例化委托，就会发生此错误。委托的参数类型必须与您打算分配给委托的函数匹配。

示例

下面的示例生成 CS0410：

```
// CS0410.cs
// compile with: /langversion:ISO-1

class Test
{
    delegate void D(double d );
    static void F(int i) { }

    static void Main()
    {
        D d = new D(F); // CS0410
    }
}
```

编译器错误 CS0411

错误消息

无法从用法推断出方法“method”的类型参数。请尝试显式指定类型参数。

如果您调用某个泛型方法但没有显式提供类型参数，导致编译器无法推断出哪些是要使用的类型参数，则会发生此错误。若要避免此错误，请将要使用的类型参数添加到尖括号中。

示例

下面的示例生成 CS0411：

```
// CS0411.cs
class C
{
    void G<T>()
    {
    }

    public static void Main()
    {
        G(); // CS0411
        // Try this instead:
        // G<int>();
    }
}
```

其他可能的错误情况包括当参数为 `null` 时(没有类型信息)：

```
// CS0411b.cs
class C
{
    public void F<T>(T t) where T : C
    {
    }

    public static void Main()
    {
        C c = new C();
        c.F(null); // CS0411
    }
}
```

另一种可能的情况是当转换涉及多个参数之一时：

```
// CS0411c.cs

class C
{
    void F<T>(T t1, T t2)
    {
    }

    public static void Main()
    {
        C c = new C();
        c.F(1, 2L); // CS0411 -- is T int or long?
    }
}
```

编译器错误 CS0412

错误消息

"generic":参数或局部变量不能与方法类型参数具有相同的名称

在泛型方法的类型参数与该方法的一个局部变量或该方法的一个参数之间存在名称冲突。若要避免此错误，请重命名任何有冲突的参数或局部变量。

示例

下面的示例生成 CS0412:

```
// CS0412.cs
using System;

class C
{
    // Parameter name is the same as method type parameter name
    public void G<T>(int T)    // CS0412
    {
    }
    public void F<T>()
    {
        // Method local variable name is the same as method type
        // parameter name
        double T = 0.0;    // CS0412
        Console.WriteLine(T);
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0413

错误消息

由于类型参数“type parameter”既没有类类型约束，也没有“class”约束，因此不能与“as”运算符一起使用

如果某个泛型类型使用了 `as` 运算符，但该泛型类型不具有类类型约束，则发生此错误。`as` 运算符只允许用于引用类型，因此必须约束类型参数以保证它不是值类型。要避免此错误，请使用类类型约束或引用类型约束。

这是因为 `as` 运算符能够返回 `null`（它不可能是值类型的值），并且类型参数必须被视为值类型，除非它是类类型约束或引用类型约束。

示例

下面的示例生成 CS0413。

```
// CS0413.cs
// compile with: /target:library
class A {}
class B : A {}

class CMain
{
    A a = null;
    public void G<T>()
    {
        a = new A();
        System.Console.WriteLine (a as T); // CS0413
    }

    // OK
    public void H<T>() where T : A
    {
        a = new A();
        System.Console.WriteLine (a as T);
    }
}
```

编译器错误 CS0415

错误消息

"IndexerName" 属性仅在不是显式接口成员声明的索引器上有效

如果您在是接口的显式实现的索引器上使用 IndexerName 属性，则将出现此错误。如果有可能，通过从索引器的声明中移除接口名称，可以避免此错误。有关更多信息，请参见 [IndexerNameAttribute 类](#)。

下面的示例生成 CS0415：

```
// CS0415.cs
using System;
using System.Runtime.CompilerServices;

public interface IA
{
    int this[int index]
    {
        get;
        set;
    }
}

public class A : IA
{
    [IndexerName("Item")] // CS0415
    int IA.this[int index]
    // Try this line instead:
    // public int this[int index]
    {
        get { return 0; }
        set { }
    }

    static void Main()
    {
    }
}
```

编译器错误 CS0416

错误消息

"type parameter": 属性参数不能使用类型参数

类型参数被用作属性参数，这是不允许的。请使用非泛型类型。

下面的示例生成 CS0416：

```
// CS0416.cs
public class MyAttribute : System.Attribute
{
    public MyAttribute(System.Type t)
    {
    }
}

class G<T>
{
    [MyAttribute(typeof(G<T>))] // CS0416
    public void F()
    {
    }
}
```

编译器错误 CS0417

错误消息

"identifier": 创建变量类型的实例时无法提供参数

如果对类型参数上的 new 运算符的调用带有参数，则会发生此错误。可以使用未知参数类型上的 new 运算符调用的唯一构造函数是不带参数的构造函数。如果需要调用另一个构造函数，请考虑使用类类型约束或接口约束。

示例

下面的示例生成 CS0417:

```
// CS0417
class C<T> where T : new()
{
    T type = new T(1);    // CS0417
}
```

编译器错误 CS0418

错误消息

"class name": 抽象类不能是密封的或静态的

除非从抽象类继承，否则抽象类不能用于创建对象，所以它成为密封的没有意义。抽象类成为静态也没有意义；设计抽象类是为了支持将抽象类用作基类的对象层次结构。

示例

下面的示例生成 CS0418：

```
// CS0418.cs
public abstract sealed class C // CS0418
{
}

sealed static class S // CS0418
{
}

public class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0423

错误消息

由于“class”具有 ComImport 属性, “method”必须为外部的或抽象的

指定 ComImport 属性意味着类的实现将从 COM 模块导入。不能定义额外的方法。

下面的示例生成 CS0423:

```
// CS0423.cs

using System.Runtime.InteropServices;

[
    ComImport,
    Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829")
]
class ImageProperties
{
    public static void Main() // CS0423
    {
        ImageProperties i = new ImageProperties();
    }
}
```

编译器错误 CS0424

错误消息

"class": 具有 ComImport 属性的类不能指定基类

指定 [ComImportAttribute](#) 属性意味着类的实现将从 COM 模块导入。不允许将从基类继承的其他方法或字段添加到在 COM 模块中定义的实现中。

下面的示例生成 CS0424:

```
// CS0424.cs
// compile with: /target:library
using System.Runtime.InteropServices;
public class A {}

[ ComImport, Guid("7ab770c7-0e23-4d7a-8aa2-19bfad479829") ]
class B : A {} // CS0424 error
```

编译器错误 CS0425

错误消息

方法“method”的类型参数“type parameter”的约束必须和接口方法“method”的类型参数“type parameter”的约束匹配。请考虑改用显式接口实现。

如果一个虚泛型方法在派生类中被重写，并且该方法在派生类中的约束与它在基类中的约束不匹配，则会发生此错误。若要避免此错误，请确保 `where` 子句在两个声明中相同，或者显式实现接口。

示例

下面的示例生成 CS0425：

```
// CS0425.cs

class C1
{
}

class C2
{
}

interface IBase
{
    void F<ItemType>(ItemType item) where ItemType : C1;
}

class Derived : IBase
{
    public void F<ItemType>(ItemType item) where ItemType : C2 // CS0425
    {
    }
}

class CMain
{
    public static void Main()
    {
    }
}
```

约束不一定要在字面上匹配，只要约束的设置有相同的含义即可。例如，以下形式是可行的：

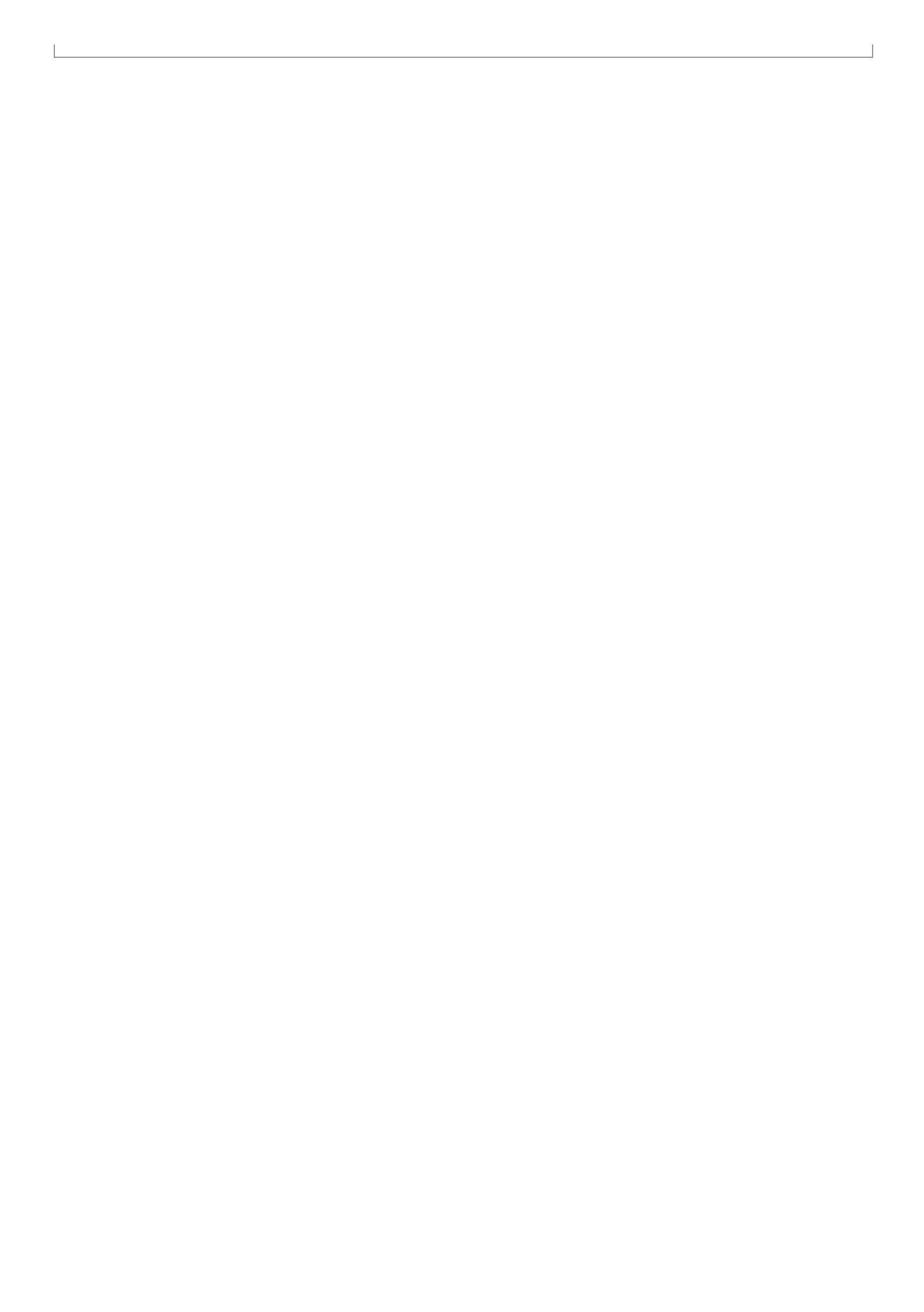
```
// CS0425b.cs

interface J<Z>
{
}

interface I<S>
{
    void F<T>(S s, T t) where T: J<S>, J<int>;
}

class C : I<int>
{
    public void F<X>(int s, X x) where X : J<int>
    {
    }

    public static void Main()
    {
    }
}
```



编译器错误 CS0426

错误消息

类型“type”中不存在类型名称“identifier”

未能在指定的类型中找到类型名称。检查所使用的名称的拼写，并验证引用的类型是否具有预期的成员。

下面的示例生成 CS0426:

```
// CS0426.cs

class C
{
}

class D
{
    public static void Main()
    {
        C.A a; // CS0426
    }
}
```

编译器错误 CS0428

错误消息

无法将方法组“Identifier”转换为非委托类型“type”。您是要调用方法吗？

将方法组转换为非委托类型时，或试图调用不带括号的方法时，会发生此错误。

示例

下面的示例生成 CS0428：

```
// CS0428.cs

delegate object Del1();
delegate int Del2();

public class C
{
    public static C Method() { return null; }
    public int Foo() { return 1; }

    public static void Main()
    {
        C c = Method; // CS0428, C is not a delegate type.
        int i = (new C()).Foo; // CS0428, int is not a delegate type.

        Del1 d1 = Method; // OK, assign to the delegate type.
        Del2 d2 = (new C()).Foo; // OK, assign to the delegate type.
        // or you might mean to invoke method
        // C c = Method();
        // int i = (new C()).Foo();
    }
}
```

编译器错误 CS0430

错误消息

/reference 选项中未指定外部别名“alias”

当遇到外部别名，而别名未在命令行上被指定为引用时，会发生此错误。若要解决 CS0430，请使用 **/reference** 进行编译。

示例

```
// CS0430_a.cs
// compile with: /target:library
public class MyClass {}
```

使用 **/reference:MyType=cs0430_a.dll** 编译来引用上面的示例中创建的 DLL 可解决此错误。下面的示例生成 CS0430。

```
// CS0430_b.cs
extern alias MyType; // CS0430
public class Test
{
    public static void Main() {}
}
```

编译器错误 CS0431

错误消息

别名“identifier”引用的是一个类型，因此不能与“::”一起使用。请改用“.”。

您将“::”与一个引用类型的别名一起使用了。若要解决此错误，请使用“.”运算符。

下面的示例生成 CS0431：

```
// CS0431.cs
using A = Outer;

public class Outer
{
    public class Inner
    {
        public static void Meth() {}
    }
}

public class MyClass
{
    public static void Main()
    {
        A::Inner.Meth();    // CS0431
        A.Inner.Meth();    // OK
    }
}
```

编译器错误 CS0432

错误消息

未找到别名“identifier”

在不是别名的标识符的右侧使用“::”时将发生此错误。若要解决此错误，请改用“.”。

下面的示例生成 CS0432:

```
// CS0432.cs
namespace A {
    public class B {
        public static void Meth() { }
    }
}

public class Test
{
    public static void Main()
    {
        A::B.Meth();    // CS0432
        // To resolve, use the following line instead:
        // A.B.Meth();
    }
}
```

编译器错误 CS0433

错误消息

类型 TypeName1 同时存在于 TypeName2 和 TypeName3 中。

在应用程序中引用的两个不同的程序集包含相同的命名空间和类型，这会产生混乱。

若要解决此错误，请使用 [/reference\(导入元数据\)\(C# 编译器选项\)](#) 编译器选项的别名功能，或者不引用您的程序集。

示例

此代码用歧义类型的第一个副本创建 DLL。

```
// CS0433_1.cs
// compile with: /target:library
namespace TypeBindConflicts
{
    public class AggPubImpAggPubImp {}
}
```

此代码用歧义类型的第二个副本创建 DLL。

```
// CS0433_2.cs
// compile with: /target:library
namespace TypeBindConflicts
{
    public class AggPubImpAggPubImp {}
}
```

下面的示例生成 CS0433。

```
// CS0433_3.cs
// compile with: /reference:cs0433_1.dll /reference:cs0433_2.dll
using TypeBindConflicts;
public class Test
{
    public static void Main()
    {
        AggPubImpAggPubImp n6 = new AggPubImpAggPubImp(); // CS0433
    }
}
```

下面的示例演示如何使用 **/reference** 编译器选项的别名功能来解决此 CS0433 错误。

```
// CS0433_4.cs
// compile with: /reference:cs0433_1.dll /reference:TypeBindConflicts=cs0433_2.dll
using TypeBindConflicts;
public class Test
{
    public static void Main()
    {
        AggPubImpAggPubImp n6 = new AggPubImpAggPubImp();
    }
}
```

编译器错误 CS0434

错误消息

NamespaceName2 中的命名空间 NamespaceName1 与 NamespaceName3 中的类型 TypeName1 冲突。

当导入的类型和导入的命名空间具有相同的完全限定名时，将出现此错误。当引用该名称时，编译器无法区分这两者。

下面的代码生成错误 CS0434。

示例

此代码用相同的完全限定名创建类型的第一个副本。

```
// CS0434_1.cs
// compile with: /t:library
namespace TypeBindConflicts
{
    namespace NsImpAggPubImp
    {
        public class X { }
    }
}
```

此代码用相同的完全限定名创建类型的第二个副本。

```
// CS0434_2.cs
// compile with: /t:library
namespace TypeBindConflicts {
    // Conflicts with another import (import2.cs).
    public class NsImpAggPubImp { }
    // Try this instead:
    // public class UniqueClassName { }
}
```

此代码用相同的完全限定名引用类型。

```
// CS0434.cs
// compile with: /r:cs0434_1.dll /r:cs0434_2.dll
using TypeBindConflicts;
public class Test
{
    public TypeBindConflicts.NsImpAggPubImp.X n2 = null; // CS0434
}
```

编译器错误 CS0438

错误消息

"module_1"中的类型"type"与"module_2"中的命名空间"namespace"冲突。

当某个源文件中的类型与其他源文件中的命名空间冲突时会发生此错误。当它们中的一个或两者都来自于某个添加的模块时，通常会发生此情况。若要解决此错误，请重命名导致此冲突的类型或命名空间。

下面的示例生成 CS0438：

首先编译此文件：

```
// CS0438_1.cs
// compile with: /target:module
public class Util
{
    public class A { }
}
```

然后，编译此文件：

```
// CS0438_2.cs
// compile with: /target:module
namespace Util
{
    public class A { }
}
```

然后，再编译此文件：

```
// CS0438_3.cs
// compile with: /addmodule:CS0438_1.netmodule /addmodule:CS0438_2.netmodule
using System;
public class Test
{
    public static void Main() {
        Console.WriteLine(typeof(Util.A));    // CS0438
    }
}
```

编译器错误 CS0439

错误消息

外部别名声明必须位于所有其他命名空间元素之前

当 **extern** 声明位于同一命名空间中某些其他元素(如 **using** 声明)之后时会发生此错误。**extern** 声明必须位于所有其他命名空间元素之前。

示例

下面的示例生成 CS0439:

```
// CS0439.cs
using System;

extern alias MyType;    // CS0439
// To resolve the error, make the extern alias the first line in the file.

public class Test
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0441

错误消息

"class":类不能同时是静态的和密封的

当您声明一个同时是静态和密封的类时，将出现此错误。静态类的密封是继承性的，所以不需要 sealed 修饰符。若要解决此问题，请仅使用一个修饰符。

下面的示例生成 CS0441：

```
// CS0441.cs
sealed static class MyClass { } // CS0441
```

编译器错误 CS0442

错误消息

"Property": 抽象属性不能有专用访问器

使用访问修饰符"private"修改抽象访问器时会发生此错误。若要解决此错误, 请使用其他访问修饰符, 或使属性成为非抽象的。

示例

下面的示例生成 CS0442:

```
// CS0442.cs
public abstract class MyClass
{
    public abstract int AbstractProperty
    {
        get;
        private set; // CS0442
        // Try this instead:
        // set;
    }
}
```

编译器错误 CS0443

错误消息

语法错误, 需要值

如果您在引用数组时未指定数组索引的值, 将发生此错误。

示例

下面的代码生成 CS0443。

```
// CS0443.cs
using System;
class MyClass
{
    public static void Main()
    {
        int[,] x = new int[1,5];
        if (x[] == 5) {} // CS0443
        // if (x[0, 0] == 5) {}
    }
}
```

编译器错误 CS0445

错误消息

无法修改取消装箱转换的结果

取消装箱转换的结果是一个临时变量，编译器禁止您修改这样的变量，因为当临时变量消失时，任何修改也随之消失。若要修复此错误，请使用一个值类型来存储中间表达式。然后您可以给值类型赋值。

下面的代码生成 CS0455：

```
// CS0445.cs
public struct Point
{
    public int x;
    public static void SetX(object obj, int x)
    {
        ((Point)obj).x = x; // CS0445
    }
}
class UnboxingTest{public static void Main(){}}
```

编译器错误 CS0446

错误消息

Foreach 不能在“方法或委托”上运行。您是要调用“方法或委托”吗？

指定一个没有括号的方法或指定一个在 **foreach** 语句部分(通常放入集合类)没有括号的匿名方法会导致此错误。请注意，如果方法返回一个集合类，则将该方法调用放在那个位置是有效的，尽管这不常见。

示例

下面的代码将生成 CS0446。

```
// CS0446.cs
using System;
class Tester
{
    static void Main()
    {
        int[] intArray = new int[5];
        foreach (int i in M) { } // CS0446
    }
    static void M() { }
}
```

编译器错误 CS0447

错误消息

属性不能用于类型变量，只能用于类型参数

将属性应用于出现在调用语句中的类型变量时会发生此错误。将属性应用于类或方法声明语句中的类型参数是可以接受的，如下所示：

```
class C<[some attribute] T> {...}
```

下面这行代码将生成此错误。假定在上一行代码中定义的类 C 有一个名为 MyStaticMethod 的静态方法。

```
C<[some attribute] T>.MyStaticMethod();
```

示例

下面的代码生成错误 CS0447。

```
// CS0447.cs
using System;
namespace Test41
{
    public interface I<A>
    {
        void Meth<B>();
    }
    public class B : I<int>
    {
        void I<[Test] int>.Meth<X>() { } // CS0447
    }
}
```

编译器错误 CS0448

错误消息

++ 或 -- 运算符的返回类型必须是包含类型或者是从包含类型派生的类型。

当您重写 ++ 或 -- 运算符时，它们必须返回与包含类型相同的类型，或者返回从包含类型派生的类型。

示例

下面的示例生成 CS0448。

```
// CS0448.cs
class C5
{
    public static int operator ++(C5 c) { return null; }    // CS0448
    public static C5 operator --(C5 c) { return null; }    // OK
    public static void Main() {}
}
```

下面的示例生成 CS0448。

```
// CS0448_b.cs
public struct S
{
    public static S? operator ++(S s) { return new S(); }    // CS0448
    public static S? operator --(S s) { return new S(); }    // CS0448
}

public struct T
{
    // OK
    public static T operator --(T t) { return new T(); }
    public static T operator ++(T t) { return new T(); }

    public static T? operator --(T? t) { return new T(); }
    public static T? operator ++(T? t) { return new T(); }

    public static void Main() {}
}
```

编译器错误 CS0449

错误消息

"class"或"struct"约束必须在其他任何约束之前

对泛型类型或方法的类型参数的约束必须以某一指定的顺序发生:**class** 或 **struct** 必须是第一个(如果存在), 然后是任何接口约束, 最后是任何构造函数约束。此错误是由于 **class** 或 **struct** 约束没有先出现所致。要解决此错误, 请重新排列约束子句。

示例

下面的示例生成 CS0449。

```
// CS0449.cs
// compile with: /target:library
interface I {}
public class C4
{
    public void F1<T>() where T : class, struct, I {}    // CS0449
    public void F2<T>() where T : I, struct {}    // CS0449
    public void F3<T>() where T : I, class {}    // CS0449

    // OK
    public void F4<T>() where T : class {}
    public void F5<T>() where T : struct {}
    public void F6<T>() where T : I {}
}
```

编译器错误 CS0450

错误消息

"Type Parameter Name": 不能同时指定约束类和"class"或"struct"约束

如果类型参数受 struct 类型约束的约束，则它也受特定 class 类型的约束在逻辑上是矛盾的，因为 struct 和 class 属于互相排斥的类别。如果类型参数受特定 class 类型约束的约束，则按照定义它应该受 class 类型约束的约束，因此指定 class 类型约束是多余的。

示例

```
// CS0450.cs
// compile with: /t:library
public class GenericsErrors
{
    public class B { }
    public class G3<T> where T : struct, B { } // CS0450
    // To resolve, use the following line instead:
    // public class G3<T> where T : B { }
}
```

编译器错误 CS0451

错误消息

"new()"约束不能与"struct"约束一起使用

当指定泛型类型的约束时, **new()** 约束只能与类类型约束、接口类型约束、引用类型约束和类型参数约束一起使用, 而不能与值类型约束一起使用。

示例

下面的示例生成 CS0451。

```
// CS0451.cs
using System;
public class C4
{
    public void F4<T>() where T : struct, new() {} // CS0451
}

// OK
public class C5
{
    public void F5<T>() where T : struct {}
}

public class C6
{
    public void F6<T>() where T : new() {}
}
```

编译器错误 CS0452

错误消息

类型“type name”必须是引用类型才能用作泛型类型或方法“identifier of generic”中的参数“parameter name”

当您将值类型(如 **struct** 或 **int**)作为参数传递给具有引用类型约束的泛型类型或方法时发生此错误。

示例

以下代码生成错误 CS0452。

```
// CS0452.cs
using System;
public class BaseClass<S> where S : class { }
public class Derived1 : BaseClass<int> { } // CS0452
public class Derived2<S> : BaseClass<S> where S : struct { } // CS0452
```

编译器错误 CS0453

错误消息

类型“Type Name”必须是不可为空的值类型才能用作泛型类型中的参数“Parameter Name”或方法“Generic Identifier”

如果您在实例化具有 **value** 约束的泛型类型或方法时使用非值类型参数，就会发生此错误。使用可为空的值类型参数时也可能发生此错误。请参见下面的代码示例的最后两行。

示例

下面的代码生成此错误。

```
// CS0453.cs
using System;
public class HV<S> where S : struct { }
public class H1 : HV<string> { } // CS0453
public class H2 : HV<H1> { } // CS0453
public class H3<S> : HV<S> where S : class { } // CS0453
public class H4 : HV<int?> { } // CS0453
public class H5 : HV<Nullable<Nullable<int>>> { } // CS0453
```

编译器错误 CS0454

错误消息

涉及“Type Parameter 1”和“Type Parameter 2”的循环约束依赖项

由于有时一个类型参数引用另一个，而第二个参数反过来又引用第一个，因此会出现此错误。要修复此错误，请移除其中一个约束以中断循环依赖项。请注意，循环约束依赖项可以是间接的。

示例

下面的代码生成错误 CS0454。

```
// CS0554
using System;
public class GenericsErrors
{
    public class G4<T> where T : T { } // CS0454
}
```

下面的示例显示了一个在两个类型约束之间的循环依赖项。

```
public class Gen<T,U> where T : U where U : T // CS0454
{ }
```

编译器错误 CS0455

错误消息

类型参数“Type Parameter Name”继承有冲突的约束“Constraint Name 1”和“Constraint Name 2”

产生此错误的两种常见情况是：设置约束，使类型参数从两个不相关的类派生，或者使类型参数从类类型或引用类型约束以及 **struct** 类型或值类型约束派生。要解决此错误，请移除您的继承层次结构中的冲突。

示例

以下代码生成错误 CS0455。

```
// CS0455.cs
using System;

public class GenericsErrors {
    public class B { }
    public class B2 { }
    public class G6<T> where T : B { public class N<U> where U : B2, T { } } // CS0455
}
```

编译器错误 CS0456

错误消息

类型参数“Type Parameter Name 1”具有“struct”约束，因此“Type Parameter Name 1”不能用作“Type Parameter Name 2”的约束。

值类型约束被隐式密封，因此那些约束不能用作另一个类型参数的约束。这是因为无法重写值类型。要解决此错误，请直接在第二个类型参数上设置值类型约束，而不是通过第一个类型参数间接这样做。

示例

下面的示例生成 CS0456。

```
// CS0456.cs
// compile with: /target:library
public class GenericsErrors
{
    public class G5<T> where T : struct
    {
        public class N<U> where U : T {} // CS0456
        public class N2<U> where U : struct {} // OK
    }
}
```

编译器错误 CS0457

错误消息

当从“type name 1”转换为“type name 2”时，用户定义的转换“Conversion method name 1”和“Conversion method name 2”不明确。

有两个转换方法可以使用，编译器无法确定使用哪一个。

可能导致此错误的一种情况如下：

- 您想从类 A 转换到类 B，其中 A 和 B 不相关。
- A 从 A0 派生，有一个从 A0 转换到 B 的方法。
- B 有一个子类 B1，有一个从 A 转换到 B1 的方法。

编译器将认为这两个转换方法是等价的，因为第一个转换方法提供了最好的目标类型，而第二个转换方法提供了最好的源类型。因为编译器无法选择，所以将生成此错误。要解决此问题，请编写一个从 A 转换到 B 的新的显式方法。

另一个导致此错误的情况是有两个从 A 转换到 B 的方法。要修复此错误，请指定显式的强制转换。

示例

下面的示例生成 CS0457。

```
// CS0457.cs
using System;
public class A { }

public class G0 { }
public class G1<R> : G0 { }

public class H0 {
    public static implicit operator G0(H0 h) {
        return new G0();
    }
}
public class H1<R> : H0 {
    public static implicit operator G1<R>(H1<R> h) {
        return new G1<R>();
    }
}

public class Test
{
    public static void F0(G0 g) { }
    public static void Main()
    {
        H1<A> h1a = new H1<A>();
        F0(h1a); // CS0457
    }
}
```

编译器错误 CS0459

错误消息

不能采用只读局部变量的地址

在 C# 语言中有三种生成只读局部变量的常见情况:**foreach**、**using** 和 **fixed**。在每一种情况下，您都不能写入只读局部变量或者采用它的地址。当编译器意识到您正在试图采用只读局部变量的地址时，将产生此错误。

示例

当您试图在 **foreach** 循环和 **fixed** 语句块中采用只读局部变量的地址时，下面的示例将生成 CS0459。

```
// CS0459.cs
// compile with: /unsafe

class A
{
    public unsafe void M1()
    {
        int[] ints = new int[] { 1, 2, 3 };
        foreach (int i in ints)
        {
            int *j = &i;    // CS0459
        }

        fixed (int *i = &_i)
        {
            int **j = &i;    // CS0459
        }
    }

    private int _i = 0;
}
```

编译器错误 CS0460

错误消息

重写和显式接口实现方法的约束是从基方法继承的，因此不能直接指定这些约束

当作为派生类一部分的泛型方法重写基类中的方法时，不能对重写方法指定约束。派生类中的重写方法从基类的方法中继承它的约束。

示例

下面的示例生成 CS0460。

```
// CS0460.cs
// compile with: /target:library
class BaseClass
{
    BaseClass() { }
}

interface I
{
    void F1<T>() where T : BaseClass;
    void F2<T>() where T : struct;
    void F3<T>() where T : BaseClass;
}

class ExpImpl : I
{
    void I.F1<T>() where T : BaseClass {} // CS0460
    void I.F2<T>() where T : class {} // CS0460
}
```

编译器错误 CS0462

错误消息

继承的成员“member1”和“member2”在类型“type”中具有相同的签名，因此不能重写这些成员

此错误是由于引入泛型而引起的。正常情况下，类中的方法不能有两个具有相同签名的版本。但是对于泛型，如果泛型方法使用某个特定的类型实例化，则可以指定一个可能与另一个方法重复的泛型方法。

示例

当实例化 `C<int>` 时，将为方法 `F` 创建两个具有相同签名的版本，所以类 `D` 中的重写无法确定对哪个方法应用重写。

下面的示例生成 CS0462。

```
// CS0462.cs
// compile with: /target:library
class C<T>
{
    public virtual void F(T t) {}
    public virtual void F(int t) {}
}

class D : C<int>
{
    public override void F(int t) {} // CS0462
}
```

编译器错误 CS0463

错误消息

计算十进制常数表达式失败，错误为：“error”

当十进制常数表达式在编译时溢出时会发生此错误。

溢出错误通常是在运行时发生。在本例中，您是这样定义常数表达式的：编译器可以计算结果并且知道可能会发生溢出。

示例

下面的代码生成错误 CS0463。

编译器错误 CS0466

错误消息

由于“method2”不具有 params 参数，因此“method1”也不应具有该参数

如果实现的接口不使用 **params** 参数，则不能对类成员使用该参数。

示例

下面的示例生成 CS0466。

```
// CS0466.cs
interface I
{
    void F1(params int[] a);
    void F2(int[] a);
}

class C : I
{
    void I.F1(params int[] a) {}
    void I.F2(params int[] a) {} // CS0466
    void I.F2(int[] a) {} // OK

    public static void Main()
    {
        I i = (I) new C();

        i.F1(new int[] {1, 2});
        i.F2(new int[] {1, 2});
    }
}
```

编译器错误 CS0468

错误消息

类型“type1”和类型“type2”之间存在多义性

编译程序集时，如果程序集中有两个类型具有相同的完全限定名，则会生成此错误。如果两个类型都位于添加的模块中，或者一个位于添加的模块中，一个位于源代码中，则可能出现此错误。

编译器错误 CS0470

错误消息

方法“method”无法实现类型“type”的接口访问器“accessor”。请使用显式接口实现。

访问器尝试实现接口时会生成此错误。必须使用显式接口实现。

示例

下面的示例生成 CS0470。

```
// CS0470.cs
// compile with: /target:library

interface I
{
    int P { get; }
}

class MyClass : I
{
    public int get_P() { return 0; }    // CS0470
    public int P2 { get { return 0; } }   // OK
}
```

编译器错误 CS0471

错误消息

变量“variable”不是泛型方法。如果原打算使用表达式列表，请用括号将 < 表达式括起来。

如果代码中包含的表达式列表不带括号，则会生成此错误。

示例

下面的示例生成 CS0471。

```
// CS0471.cs
// compile with: /t:library
class Test
{
    public void F(bool x, bool y) {}
    public void F1()
    {
        int a = 1, b = 2, c = 3;
        F(a<b, c>(3));      // CS0471
        // To resolve, try the following instead:
        // F((a<b), c>(3));
    }
}
```

编译器错误 CS0500

错误消息

"class member"无法声明主体, 因为它被标记为 abstract

abstract 方法不能包含自己的实现。

下面的示例生成 CS0500:

```
// CS0500.cs
namespace x
{
    abstract public class clx
    {
        abstract public void f(){} // CS0500
        // try the following line instead
        // abstract public void f();
    }

    public class cly
    {
        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0501

错误消息

"member function" 必须声明主体, 因为它未标记为 abstract 或 extern

非抽象方法必须有实现。

下面的示例生成 CS0501:

```
// CS0501.cs
// compile with: /target:library
public class Clx
{
    public void f(); // CS0501 declared not defined
    public void g() {} // OK
}
```

编译器错误 CS0502

错误消息

"member"不能既是抽象的又是密封的

一个类不能同时是 `abstract` 和 `sealed` 的。

下面的示例生成 CS0502:

```
// CS0502.cs
public class B
{
    abstract public void F();
}

public class C : B
{
    abstract sealed override public void F()    // CS0502
    {
    }
}

public class CMain
{
    public static void Main()
    { }
}
```

编译器错误 CS0503

错误消息

抽象方法“method”不能标记为 virtual

将成员方法既标记为 `abstract` 又标记为 `virtual` 是多余的，因为 **abstract** 暗指 **virtual**。

下面的示例生成 CS0503：

```
// CS0503.cs
namespace x
{
    abstract public class clx
    {
        abstract virtual public void f();    // CS0503
    }
}
```

编译器错误 CS0504

错误消息

常数“variable”无法标记为 static

如果变量是 `const`, 它也就是 `static`。如果需要一个既是 `const` 又是 `static` 的变量, 只需将该变量声明为 `const`; 如果需要的只是 `static` 变量, 只需将其标记为 `static`。

下面的示例生成 CS0504:

```
// CS0504.cs
namespace x
{
    abstract public class clx
    {
        static const int i = 0;    // CS0504, cannot be both static and const
        abstract public void f();
    }
}
```

编译器错误 CS0505

错误消息

"member1": 无法重写, 因为"member2"不是函数

类声明试图重写基类中非方法的成员。重写必须与成员类型相匹配。如果希望使用与基类中的方法同名的方法, 请在基类的方法声明中使用 [new](#)(而不使用 [override](#))。

下面的示例生成 CS0505:

```
// CS0505.cs
// compile with: /target:library
public class clx
{
    public int i;
}

public class cly : clx
{
    public override int i() { return 0; }    // CS0505
}
```

编译器错误 CS0506

错误消息

"function1": 无法重写继承成员 "function2"，因为它未标记为 "virtual"、"abstract" 或 "override"

将未显式标记为 **virtual**、**abstract** 或 **override** 的方法进行了重写。

下面的示例生成 CS0506:

```
// CS0506.cs
namespace MyNameSpace
{
    abstract public class ClassX
    {
        public int i = 0;

        public int f()
        {
            return 0;
        }
        // Try the following definition for f() instead:
        // abstract public int f();
    }

    public class ClassY : ClassX
    {
        public override int f() // CS0506
        {
            return 0;
        }

        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0507

错误消息

"function1": 当重写"access"继承成员"function2"时, 无法更改访问修饰符

试图在方法重写中更改访问规范。

示例

下面的示例生成 CS0507。

```
// CS0507.cs
abstract public class clx
{
    virtual protected void f() {}

public class cly : clx
{
    public override void f() {} // CS0507
    public static void Main() {}
}
```

当类试图重写在引用的元数据中定义的标记为 **protected internal** 的方法时, 也会发生 CS0507。在这种情况下, 重写方法应标记为 **protected**。

```
// CS0507_b.cs
// compile with: /target:library
abstract public class clx
{
    virtual protected internal void f() {}
}
```

下面的示例生成 CS0507。

```
// CS0507_c.cs
// compile with: /reference:cs0507_b.dll
public class cly : clx
{
    protected internal override void f() {} // CS0507
    // try the following line instead
    // protected override void f() {} // OK

    public static void Main() {}
}
```

编译器错误 CS0508

错误消息

"Type 1": 返回类型必须是"Type 2"才能与重写成员"Member Name"匹配

试图在方法重写中更改返回类型。若要解决此错误, 请确保两个方法都声明相同的返回类型。

示例

下面的示例生成 CS0508。

```
// CS0508.cs
// compile with: /target:library
abstract public class Clx
{
    public int i = 0;
    // Return type is int.
    abstract public int F();
}

public class Cly : Clx
{
    public override double F()
    {
        return 0.0;    // CS0508
    }
}
```

编译器错误 CS0509

错误消息

"class1": 无法从密封类型"class2"派生

`sealed` 类不能用作 `base` 类。默认情况下，结构被密封。

下面的示例生成 CS0509:

```
// CS0509.cs
// compile with: /target:library
sealed public class clx {}
public class cly : clx {} // CS0509
```

编译器错误 CS0513

错误消息

"function"是抽象的, 但它包含在非抽象类"class"中

方法不能是非抽象类的 `abstract` 成员。

下面的示例生成 CS0513:

```
// CS0513.cs
namespace x
{
    public class clx
    {
        abstract public void f(); // CS0513, abstract member of nonabstract class
    }
}
```

编译器错误 CS0514

错误消息

"constructor": 静态构造函数不能具有显式的"this"或"base"构造函数调用

在静态构造函数中不允许调用 **this**, 因为在创建类的任何实例之前都会自动调用静态构造函数。另外, 静态构造函数是不可继承的, 而且不能被直接调用。

有关更多信息, 请参见 [this\(C# 参考\)](#) 和 [base\(C# 参考\)](#)。

示例

下面的示例生成 CS0514:

```
// CS0514.cs
class A
{
    static A() : base(0) // CS0514
    {
    }

    public A(object o)
    {
    }
}

class B
{
    static B() : this(null) // CS0514
    {
    }

    public B(object o)
    {
    }
}
```

编译器错误 CS0515

错误消息

"function": 静态构造函数中不允许出现访问修饰符

静态构造函数不能有[访问修饰符](#)。

示例

下面的示例生成 CS0515:

```
// CS0515.cs
public class Clx
{
    public static void Main()
    {
    }
}

public class Clz
{
    public static Clz()    // CS0515, remove public keyword
    {
    }
}
```

编译器错误 CS0516

错误消息

构造函数“constructor”不能调用自身

程序不能递归调用构造函数。

下面的示例生成 CS0516:

```
// CS0516.cs
namespace x
{
    public class clx
    {
        public clx() : this() // CS0516, delete "this()"
        {
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0517

错误消息

"class"没有基类, 无法调用基构造函数

仅当 .NET Framework 公共语言运行库编译对象类的源代码而该类是唯一没有基类的类时才发生 CS0517。

编译器错误 CS0518

错误消息

预定义的类型“type”未定义或未导入

引起该问题的主要原因是项目没有导入 mscorelib.dll，该文件定义整个 System 命名空间。这可能由以下原因之一导致：

- 指定了命令行编译器的 [/nostdlib](#) 选项。[/nostdlib](#) 选项将禁止导入 mscorelib.dll。如果想要定义或创建用户特定的 System 命名空间，请使用该选项。
- 引用了错误的 mscorelib.dll。
- Visual Studio .NET 或 .NET Framework 公共语言运行库安装存在损坏。
- 早期安装所剩的组件与最新安装的其余组件不兼容。

要解决此问题，请采用以下操作之一：

- 不指定命令行编译器的 [/nostdlib](#) 选项。
- 确保项目引用正确的 mscorelib.dll。
- 重新安装 .NET Framework 公共语言运行库(若以上解决方案均未能解决此问题)。

编译器错误 CS0520

错误消息

预定义类型“name”未正确声明

未找到编译器所需的一个或多个文件。重新安装 .NET Framework 公共语言运行库。

编译器错误 CS0522

错误消息

"constructor": 结构无法调用基类构造函数

`struct` 不能调用基类构造函数; 请移除对基类构造函数的调用。

下面的示例生成 CS0522:

```
// CS0522.cs
public class clx
{
    public clx(int i)
    {
    }

    public static void Main()
    {
    }
}

public struct cly
{
    public cly(int i):base(0) // CS0522
    // try the following line instead
    // public cly(int i)
    {
    }
}
```

编译器错误 CS0523

错误消息

"struct1"类型的结构成员"struct2 field"在结构布局中导致循环

两个结构的定义包含递归引用。更改 `struct` 定义以便每个结构都不会在另一结构上定义自身。此限制仅适用于结构，因为结构是值类型。如果使用递归引用，请将类型声明为类。

下面的示例生成 CS0523：

```
// CS0523.cs
// compile with: /target:library
struct RecursiveLayoutStruct1
{
    public RecursiveLayoutStruct2 field;
}

struct RecursiveLayoutStruct2
{
    public RecursiveLayoutStruct1 field;    // CS0523
}
```

编译器错误 CS0524

错误消息

"type": 接口不能声明类型。

接口不能包含用户定义的类型，只能包含方法和属性。

示例

下面的示例生成 CS0524：

```
// CS0524.cs
public interface Clx
{
    public class Cly    // CS0524, delete user-defined type
    {
    }
}
```

编译器错误 CS0525

错误消息

接口不能包含字段

`interface` 可以包含方法和属性，但不能包含字段。

下面的示例生成 CS0525：

```
// CS0525.cs
namespace x
{
    public interface clx
    {
        public int i;    // CS0525
    }
}
```

编译器错误 CS0526

错误消息

接口不能包含构造函数

不能为 [interface](#) 定义构造函数。如果方法与类同名且没有返回类型，则将其视为构造函数。

下面的示例生成 CS0526:

```
// CS0526.cs
namespace x
{
    public interface clx
    {
        public clx() // CS0526
    }
}

public class cly
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0527

错误消息

接口列表中的类型“type”不是接口

对于 [struct](#) 或 [interface](#), 从其他接口而不是其他任何类型继承是可能的。

下面的示例生成 CS0527:

```
// CS0527.cs
// compile with: /target:library
public struct clk : int {}    // CS0527 int not an interface
```

编译器错误 CS0528

错误消息

"interface"已经在接口列表中列出

接口继承列表中有重复接口。[interface](#) 在继承列表中只能指定一次。

下面的示例生成 CS0528:

```
// CS0528.cs
namespace x
{
    public interface a
    {
    }

    public class b : a, a // CS0528
    {
        public void Main()
        {
        }
    }
}
```

编译器错误 CS0529

错误消息

继承接口“interface1”在“interface2”的接口层次结构中导致一个循环

[interface](#) 的继承列表包括对自身的直接或间接引用。接口不能从自身继承。

下面的示例生成 CS0529:

```
// CS0529.cs
namespace x
{
    public interface a
    {
    }

    public interface b : a, c
    {
    }

    public interface c : b // CS0529, b inherits from c
    {
    }
}
```

编译器错误 CS0531

错误消息

"member": 接口成员不能有定义

[interface](#) 中声明的方法必须从该接口继承的类中实现而不是在接口自身中实现。

下面的示例生成 CS0531:

```
// CS0531.cs
namespace x
{
    public interface clx
    {
        int xclx() // CS0531, cannot define xclx
        // Try the following declaration instead:
        // int xclx();
        {
            return 0;
        }
    }

    public class cly
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0533

错误消息

"derived-class member"隐藏继承的抽象成员"base-class member"

基 [class](#) 方法被隐藏。请检查声明语法以查看它是否正确。

有关更多信息, 请参见 [base](#)。

下面的示例生成 CS0533:

```
// CS0533.cs
namespace x
{
    abstract public class a
    {
        abstract public void f();
    }

    abstract public class b : a
    {
        new abstract public void f(); // CS0533
        // try the following lines instead
        // override public void f()
        // {
        // }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0534

错误消息

"function1"不会实现继承的抽象成员"function2"

一个类应该实现基类中的所有 `abstract` 成员，除非该类也是抽象的。

下面的示例生成 CS0534:

```
// CS0534.cs
namespace x
{
    abstract public class clx
    {
        public abstract void f();
    }

    public class cly : clx // CS0534, no override for clx::f
    {
        // uncomment the following sample override to resolve CS0534
        // override public void f()
        // {
        // }

        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0535

错误消息

"class"不会实现接口成员"member"

class 从 interface 派生, 但该类未实现该接口的一个或多个成员。类必须实现它从中派生的接口的所有成员, 否则必须声明为 abstract。

示例

下面的示例生成 CS0535。

```
// CS0535.cs
public interface A
{
    void F();
}

public class B : A {} // CS0535 A::F is not implemented

// OK
public class C : A {
    public void F() {}
    public static void Main() {}
}
```

下面的示例生成 CS0535。

```
// CS0535_b.cs
using System;
class C : IDisposable {} // CS0535

// OK
class D : IDisposable {
    void IDisposable.Dispose() {}
    public void Dispose() {}

    static void Main() {
        using (D d = new D()) {}
    }
}
```

编译器错误 CS0536

错误消息

"class"不会实现接口成员"interface member"。"class member"是静态的、非公共的, 或者有错误的返回类型

" **class** "不实现接口成员 "**member1** ." **member2** "是静态、非公共的, 或者有错误的返回类型。

编译器未检测到 [interface](#) 成员实现。可能存在几乎实现该接口成员的声明。请检查接口成员声明中是否存在下列语法错误:

- **public** 关键字被省略。
- 返回类型不匹配。
- 存在 **static** 关键字。

下面的示例生成 CS0536:

```
// CS0536.cs
public interface a
{
    void f();
}

public class b : a
{
    public static int f() // CS0536
    // try the following line instead
    // public void f()
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0537

错误消息

类 System.Object 不能有基类也不能实现接口

重新生成 [System](#) 类库时, 以及在 [Object](#) 从其他类派生时, 将发生 CS0537。您不大可能会遇到此错误。如果您确实遇到此错误, 请不要从类或接口派生 **Object**: 它是整个 .NET Framework 类层次结构的根, 因此不从任何其他类或接口派生。

编译器错误 CS0538

错误消息

显式接口声明中的“name”不是接口

试图显式声明 [interface](#), 但未指定接口。

下面的示例生成 CS0538:

```
// CS0538.cs
interface MyIFace
{
    void F();
}

public class MyClass
{
    public void G()
    {
    }
}

class C: MyIFace
{
    void MyIFace.F()
    {

    }

    void MyClass.G() // CS0538, MyClass not an interface
    {
    }
}
```

编译器错误 CS0539

错误消息

显式接口声明中的“member”不是接口成员

试图显式声明不存在的 [interface](#) 成员。应删除声明或对其进行更改，以便它引用有效的接口成员。

下面的示例生成 CS0539：

```
// CS0539.cs
namespace x
{
    interface I
    {
        void m();
    }

    public class clx : I
    {
        void I.x() // CS0539
        {
        }

        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0540

错误消息

"interface member": 包含类型不实现接口 "interface"

试图实现不是从 [interface](#) 派生的 [class](#) 中的接口成员。应删除接口成员的实现，或将接口添加到类的基类列表中。

示例

下面的示例生成 CS0540。

```
// CS0540.cs
interface I
{
    void m();
}

public class Clx
{
    void I.m() {} // CS0540
}

// OK
public class Cly : I
{
    void I.m() {}
    public static void Main() {}
}
```

下面的示例生成 CS0540。

```
// CS0540_b.cs
using System;
class C {
    void IDisposable.Dispose() {} // CS0540
}

class D : IDisposable {
    void IDisposable.Dispose() {}
    public void Dispose() {}

    static void Main() {
        using (D d = new D()) {}
    }
}
```

编译器错误 CS0541

错误消息

"declaration": 显式接口声明只能在类或结构中声明

在 `class` 或 `struct` 之外找到了显式的 `interface` 声明。

下面的示例生成 CS0541:

```
// CS0541.cs
namespace x
{
    interface IFace
    {
        void F();
    }

    interface IFace2 : IFace
    {
        void IFace.F(); // CS0541
    }
}
```

编译器错误 CS0542

错误消息

"user-defined type": 成员名称不能与它们的封闭类型相同

某一名称在同一构造中多次使用。造成此错误的原因可能是因疏忽而将返回类型放在了构造函数中。

下面的示例生成 CS0542:

```
// CS0542.cs
class F
{
    // Remove void from F() to resolve the problem.
    void F()    // CS0542, same name as the class
    {
    }
}

class MyClass
{
    public static void Main()
    {
    }
}
```

如果您的类被命名为“Item”，并且有一个声明为 `this` 的索引器，您可能会遇到此错误。在发出的代码中为默认索引器给出的名称是“Item”，从而导致冲突。

```
// CS0542b.cs
class Item
{
    public int this[int i]    // CS0542
    {
        get
        {
            return 0;
        }
    }
}

class CMain
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0543

错误消息

"enumeration": 枚举数值太大, 不能适应它的类型

赋给 `enumeration` 中元素的值在数据类型范围之外。

下面的示例生成 CS0543:

```
// CS0543.cs
namespace x
{
    enum I : byte
    {a = 255, b, c}    // CS0543
    public class clk
    {
        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS0544

错误消息

"property override": 无法重写, 因为"non-property"不是属性

试图将非属性数据类型重写为属性, 这是不允许的。

下面的示例生成 CS0544:

```
// CS0544.cs
// compile with: /target:library
public class a
{
    public int i;
}

public class b : a
{
    public override int i {    // CS0544
        // try the following line instead
        // public new int i {
            get
            {
                return 0;
            }
        }
}
```

编译器错误 CS0545

错误消息

"function": 无法重写, 因为"property"没有可重写的 get 访问器

试图定义属性访问器的重写, 而此时基类没有这样的重写定义。可以通过下面的方法解决该错误:

- 在基类中添加 **set** 访问器。
- 从派生类中移除 **set** 访问器。
- 通过在派生类的属性中添加 **new** 关键字来隐藏基类属性。
- 生成基类属性 **virtual**。

有关更多信息, 请参见[使用属性\(C# 编程指南\)](#)和[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0545。

```
// CS0545.cs
// compile with: /target:library
// CS0545
public class a
{
    public virtual int i
    {
        set {}

        // Uncomment the following line to resolve.
        // get { return 0; }
    }
}

public class b : a
{
    public override int i
    {
        get { return 0; }
        set {}    // OK
    }
}
```

编译器错误 CS0546

错误消息

"accessor": 无法重写, 因为“property”没有可重写的 set 访问器

试图对属性的访问器方法之一进行重写时失败, 因为访问器不能重写。可以通过下面的方法解决该错误:

- 在基类中添加 **set** 访问器。
- 从派生类中移除 **set** 访问器。
- 通过在派生类的属性中添加 **new** 关键字来隐藏基类属性。
- 生成基类属性 **virtual**。

有关更多信息, 请参见[属性声明](#)和[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0546。

```
// CS0546.cs
// compile with: /target:library
public class a
{
    public virtual int i
    {
        get
        {
            return 0;
        }
    }

    public virtual int i2
    {
        get
        {
            return 0;
        }

        set {}
    }
}

public class b : a
{
    public override int i
    {
        set {} // CS0546 error no set
    }

    public override int i2
    {
        set {} // OK
    }
}
```

编译器错误 CS0547

错误消息

"property": 属性或索引器不能有 void 类型

void 作为属性的返回值无效。

有关更多信息, 请参见[属性](#)。

下面的示例生成 CS0547:

```
// CS0547.cs
public class a
{
    public void i // CS0547
    // Try the following declaration instead:
    // public int i
    {
        get
        {
            return 0;
        }
    }
}

public class b : a
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0548

错误消息

"property": 属性或索引器必须至少有一个访问器

属性必须至少有一个访问器 (get 或 set) 方法。

有关更多信息, 请参见[属性声明](#)和[使用属性 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS0548。

```
// CS0548.cs
// compile with: /target:library
public class b
{
    public int MyProp {} // CS0548

    public int MyProp2 // OK
    {
        get
        {
            return 0;
        }
        set {}
    }
}
```

编译器错误 CS0549

错误消息

"function"是密封类"class"中的新虚拟成员

密封类不能用作基类。因此，密封类中的虚方法是没用的。

下面的示例生成 CS0549：

```
// CS0549.cs
// compile with: /target:library
sealed public class MyClass
{
    virtual public void TestMethod() {}    // CS0549
    public void TestMethod2() {}    // OK
}
```

编译器错误 CS0550

错误消息

"accessor"添加了接口成员"property"中没有的访问器

派生类中属性的实现包含未在基接口中指定的访问器。

有关更多信息, 请参见[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0550。

```
// CS0550.cs
namespace x
{
    interface ii
    {
        int i
        {
            get;
            // add the following accessor to resolve this CS0550
            // set;
        }
    }

    public class a : ii
    {
        int ii.i
        {
            get
            {
                return 0;
            }
            set {}    // CS0550  no set in interface
        }

        public static void Main() {}
    }
}
```

编译器错误 CS0551

错误消息

显式接口实现“implementation”缺少访问器“accessor”

显式实现接口属性的类必须实现接口所定义的所有访问器。

有关更多信息, 请参见[属性声明和使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0551。

```
// CS0551.cs
// compile with: /target:library
interface ii
{
    int i
    {
        get;
        set;
    }
}

public class a : ii
{
    int ii.i { set {} } // CS0551

    // OK
    int ii.i
    {
        set {}
        get { return 0; }
    }
}
```

编译器错误 CS0552

错误消息

"conversion routine": 与接口之间的用户定义转换

不能创建用户定义的接口转换。如果需要转换例程，则使接口成为类来解决该错误，或者从接口派生类。

下面的示例生成 CS0552:

```
// CS0552.cs
public interface ii
{
}

public class a
{
    // delete the routine to resolve CS0552
    public static implicit operator ii(a aa) // CS0552
    {
        return new ii();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0553

错误消息

"conversion routine": 与基类之间的用户定义转换

不允许用户定义的到基类值的转换; 您不需要这样的运算符。

下面的示例生成 CS0553:

```
// CS0553.cs
namespace x
{
    public class ii
    {
    }

    public class a : ii
    {
        // delete the conversion routine to resolve CS0553
        public static implicit operator ii(a aa) // CS0553
        {
            return new ii();
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0554

错误消息

"conversion routine": 与派生类之间的用户定义转换

不允许用户定义的到派生类值的转换; 您不需要这样的运算符。

有关用户定义的转换的更多信息, 请参见 C# 语言规范的第 6 章。

下面的示例生成 CS0554:

```
// CS0554.cs
namespace x
{
    public class ii
    {
        // delete the conversion routine to resolve CS0554
        public static implicit operator ii(a aa) // CS0554
        {
            return new ii();
        }
    }

    public class a : ii
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0555

错误消息

用户定义的运算符不能采用封闭类型的对象，也不能转换成封闭类型的对象

不允许用户定义的到封闭类值的转换；您不需要这样的运算符。

下面的示例生成 CS0555：

```
// CS0555.cs
public class MyClass
{
    // delete the following operator to resolve this CS0555
    public static implicit operator MyClass(MyClass aa)    // CS0555
    {
        return new MyClass();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0556

错误消息

用户定义的转换必须是转换成封闭类型，或者从封闭类型转换

用户定义的转换例程必须转换成包含该例程的类，或者从包含该例程的类进行转换。

下面的示例生成 CS0556：

```
// CS0556.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            public static implicit operator int(byte aa) // CS0556
            // try the following line instead
            // public static implicit operator int(iii aa)
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0557

错误消息

类型“class”中有重复的用户定义转换

类中不允许有重复的转换例程。

下面的示例生成 CS0557:

```
// CS0557.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            public static implicit operator int(iii aa)
            {
                return 0;
            }

            // CS0557, delete duplicate
            public static explicit operator int(iii aa)
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0558

错误消息

用户定义的运算符“operator”必须声明为 static 和 public

static 和 **public** 访问修饰符必须在用户定义的运算符上指定。

下面的示例生成 CS0558:

```
// CS0558.cs
namespace x
{
    public class ii
    {
        public class iii
        {
            static implicit operator int(iii aa) // CS0558, add public
            {
                return 0;
            }
        }

        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0559

错误消息

`++` 或 `--` 运算符的参数类型必须是包含类型

运算符重载的方法声明必须遵循一定的准则。`++` 和 `--` 运算符要求参数的类型与重载该运算符的类型相同。

示例

下面的示例生成 CS0559:

```
// CS0559.cs
// compile with: /target:library
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }

    public static implicit operator iii(int x)
    {
        return null;
    }

    public static int operator ++(int aa) // CS0559
    // try the following line instead
    // public static iii operator ++(iii aa)
    {
        return (iii)0;
    }
}
```

下面的示例生成 CS0559。

```
// CS0559_b.cs
// compile with: /target:library
public struct S
{
    public static S operator ++(S? s) { return new S(); } // CS0559
    public static S operator --(S? s) { return new S(); } // CS0559
}

public struct T
{
    // OK
    public static T operator --(T t) { return new T(); }
    public static T operator ++(T t) { return new T(); }

    public static T? operator --(T? t) { return new T(); }
    public static T? operator ++(T? t) { return new T(); }
}
```

编译器错误 CS0562

错误消息

一元运算符的参数必须是包含类型

运算符重载的方法声明必须遵循一定的准则。有关更多信息, 请参见[可重载运算符](#)和[“运算符重载”示例](#)。

示例

下面的示例生成 CS0562:

```
// CS0562.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }

    public static implicit operator iii(int x)
    {
        return null;
    }

    public static iii operator +(int aa) // CS0562
    // try the following line instead
    // public static iii operator +(iii aa)
    {
        return (iii)0;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0563

错误消息

二元运算符的参数之一必须是包含类型

运算符重载的方法声明必须遵循一定的准则。有关更多信息, 请参见[可重载运算符](#)和[“运算符重载”示例](#)。

示例

下面的示例生成 CS0563:

```
// CS0563.cs
public class iii
{
    public static implicit operator int(iii x)
    {
        return 0;
    }
    public static implicit operator iii(int x)
    {
        return null;
    }
    public static int operator +(int aa, int bb) // CS0563
    // Use the following line instead:
    // public static int operator +(int aa, iii bb)
    {
        return 0;
    }
    public static void Main()
    {
    }
}
```

编译器错误 CS0564

错误消息

重载移位运算符的第一个操作数的类型必须是包含类型，第二个操作数的类型必须是 int 类型。

试图用错误类型的操作数重载移位运算符(<< 或 >>)。第一个操作数必须是该包含类型，并且第二个操作数必须是 int 类型的。

下面的示例生成 CS0564：

```
// CS0564.cs
using System;
class C
{
    public static int operator << (C c1, C c2) // CS0564
// To correct, change second operand to int, like so:
// public static int operator << (C c1, int c2)
    {
        return 0;
    }
    static void Main()
    {
    }
}
```

编译器错误 CS0567

错误消息

接口不能包含运算符

`interface` 定义中不允许有运算符。

下面的示例生成 CS0567:

```
// CS0567.cs
interface IA
{
    int operator +(int aa, int bb);    // CS0567
}

class Sample
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0568

错误消息

结构不能包含显式的无参数构造函数

每个 `struct` 均已有一个将对象初始化为零的默认构造函数。因此，可以为结构创建的构造函数必须带有一个或多个参数。

下面的示例生成 CS0568：

```
// CS0568.cs
public struct ClassY
{
    public int field1;
    public ClassY(){} // CS0568, cannot have no param constructor
    // Try following instead:
    // public ClassY(int i)
    // {
    //     field1 = i;
    // }
}

public class ClassX
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0569

错误消息

"method2": 无法重写 "method1", 因为该语言不支持它

当您从以另一种语言编写的基类进行派生时，并且当编译器不理解您正试图重写的方法时发生此错误。

编译器错误 CS0570

错误消息

该语言不支持“class”

当使用由另一编译器生成的导入元数据时发生此错误。您的代码试图使用编译器无法处理的类成员。

示例

下面的 C++ 程序使用了一种其他语言不能使用的属性 RequiredAttributeAttribute。

```
// CPP0570.cpp
// compile with: /clr /LD

using namespace System;
using namespace System::Runtime::CompilerServices;

namespace CS0570_Server {
    [RequiredAttributeAttribute(Int32::typeid)]
    public ref struct Scenario1 {
        int intVar;
    };

    public ref struct CS0570Class {
        Scenario1 ^ sc1_field;

        property virtual Scenario1 ^ sc1_prop {
            Scenario1 ^ get() { return sc1_field; }
        }

        Scenario1 ^ sc1_method() { return sc1_field; }
    };
}
```

下面的示例生成 CS0570。

```
// CS0570.cs
// compile with: /reference:CPP0570.dll
using System;
using CS0570_Server;

public class C {
    public static int Main() {
        CS0570Class r = new CS0570Class();
        r.sc1_field = null; // CS0570
        object o = r.sc1_prop; // CS0570
        r.sc1_method(); // CS0570
    }
}
```

编译器错误 CS0571

错误消息

"function": 无法显式调用运算符或访问器

某些运算符具有内部名称。例如, **op_Increment** 是 `++` 运算符的内部名称。不应使用或显式调用这样的方法名称。

下面的示例生成 CS0571:

```
// CS0571.cs
public class MyClass
{
    public static MyClass operator ++ (MyClass c)
    {
        return null;
    }

    public static int prop
    {
        get
        {
            return 1;
        }
        set
        {
        }
    }

    public static void Main()
    {
        op_Increment(null);    // CS0571
        // use the increment operator as follows
        // MyClass x = new MyClass();
        // x++;

        set_prop(1);          // CS0571
        // try the following line instead
        // prop = 1;
    }
}
```

编译器错误 CS0572

错误消息

"type": 无法通过表达式引用类型; 请尝试改用"path_to_type"

试图通过标识符访问类的成员, 这在此情况中是不允许的。

下面的示例生成 CS0572:

```
// CS0572.cs
using System;
class C
{
    public class Inner
    {
        public static int v = 9;
    }
}

class D : C
{
    public static void Main()
    {
        C cValue = new C();
        Console.WriteLine(cValue.Inner.v);    // CS0572
        // try the following line instead
        // Console.WriteLine(C.Inner.v);
    }
}
```

编译器错误 CS0573

错误消息

"field declaration": 结构中不能有实例字段初始值设定项

不能对 [struct](#) 的实例字段进行初始化。值类型的字段将被初始化为其默认值，而引用类型字段将被初始化为 [null](#)。

示例

下面的示例生成 CS0573：

```
// CS0573.cs
namespace x
{
    public class clx
    {
        public static void Main()
        {
        }
    }

    public struct cly
    {
        clx a = new clx(); // CS0573
        // clx a;          // OK
        int i = 7;         // CS0573
        // int i;          // OK
    }
}
```

编译器错误 CS0574

错误消息

析构函数的名称必须与类的名称匹配

析构函数的名称必须是带代字号 (~) 前缀的类名称。

下面的示例生成 CS0574:

```
// CS0574.cs
namespace x
{
    public class iii
    {
        ~iiii() // CS0574
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0575

错误消息

只有类类型才能包含析构函数

`struct` 不能包含析构函数。

下面的示例生成 CS0575:

```
// CS0575.cs
namespace x
{
    public struct iii
    {
        ~iii() // CS0575
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0576

错误消息

命名空间“namespace”包含的定义与别名“identifier”冲突

试图第二次使用同一命名空间。

示例

下面的示例生成 CS0576:

```
// CS0576.cs
using SysIO = System.IO;
public class SysIO
{
    public void MyMethod() {}
}

public class Test
{
    public static void Main()
    {
        SysIO.Stream s; // CS0576
    }
}
```

编译器错误 CS0577

错误消息

Conditional 属性在“function”上无效，因为它是构造函数、析构函数、运算符或显式接口实现

Conditional 无法应用于指定的方法。

例如，您无法对显式接口定义使用某些属性。下面的示例生成 CS0577：

```
// CS0577.cs
// compile with: /target:library
interface I
{
    void m();
}

public class MyClass : I
{
    [System.Diagnostics.Conditional("a")] // CS0577
    void I.m() {}
}
```

编译器错误 CS0578

错误消息

Conditional 属性在“function”上无效，因为其返回类型不是 void

[ConditionalAttribute](#) 不能应用于返回类型不是 **void** 的方法。其原因在于：程序的其他部分可能需要方法的任何其他返回类型。

示例

下面的示例生成 CS0578。若要解决此错误，必须删除 **ConditionalAttribute**，或者将方法的返回值更改为 **void**。

```
// CS0578.cs
// compile with: /target:library
public class MyClass
{
    [System.Diagnostics.ConditionalAttribute("a")]    // CS0578
    public int TestMethod()
    {
        return 0;
    }
}
```

编译器错误 CS0579

错误消息

重复“attribute”属性

不可能多次指定相同的属性，除非属性在其 [AttributeUsage](#) 中指定 **AllowMultiple=true**。

示例

下面的示例生成 CS0579。

```
// CS0579.cs
using System;
public class MyAttribute : Attribute
{
}

[AttributeUsage(AttributeTargets.All,AllowMultiple=true)]
public class MyAttribute2 : Attribute
{
}

public class z
{
    [MyAttribute, MyAttribute]      // CS0579
    public void zz()
    {
    }

    [MyAttribute2, MyAttribute2]    // OK
    public void zzz()
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0582

错误消息

Conditional 在接口成员上无效

ConditionalAttribute 在接口成员上无效。

下面的示例生成 CS0582:

```
// CS0582.cs
// compile with: /target:library
using System.Diagnostics;
interface MyIFace
{
    [ConditionalAttribute("DEBUG")]
    void zz();
}
```

编译器错误 CS0583

错误消息

内部编译器错误 ("address") : 可能的原因是 "culprit"

请使用 "/bugreport" 选项创建一个 bug 报告文件，并将其提交给为您指派的问题报告联系人。

使用 [/bugreport](#) 编译器选项捕获问题。

编译器错误 CS0584

错误消息

内部编译器错误: 步骤“stage”符号“symbol”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0585

错误消息

内部编译器错误: 步骤“stage”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0586

错误消息

内部编译器错误: 步骤“stage”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0587

错误消息

内部编译器错误: 步骤“stage”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0588

错误消息

内部编译器错误: 步骤“LEX”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0589

错误消息

内部编译器错误: 步骤“PARSE”

试图确定编译器是否由于无法分析意外的语法而失败。否则, 请尝试[从 Microsoft 产品支持服务获得帮助 \(Visual Studio\)](#)。

编译器错误 CS0590

错误消息

用户定义的运算符不能返回 void

用户定义的运算符旨在返回对象。

下面的示例生成 CS0590:

```
// CS0590.cs
namespace x
{
    public class a
    {
        public static void operator+(a A1, a A2)    // CS0590
        {
        }

        // try the following user-defined operator
        /*
        public static a operator+(a A1, a A2)
        {
            return A2;
        }
        */
    }

    public static int Main()
    {
        return 1;
    }
}
```

编译器错误 CS0591

错误消息

"attribute"属性的参数值无效

传递给属性的是一个无效参数或两个互斥参数。

示例

下面的示例生成 CS0591:

```
// CS0591.cs
using System;

[AttributeUsage(0)] // CS0591
class I: Attribute
{
}

public class a
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0592

错误消息

属性“attribute”在此声明类型上无效。它只在“type”声明中有效。

对声明应用了一个对它不适用的属性。

示例

下面的示例生成 CS0592:

```
// CS0592.cs
using System;

[AttributeUsage(AttributeTargets.Interface)]
public class MyAttribute : Attribute
{
}

[MyAttribute]
public class A // CS0592, MyAttribute is not valid for a class
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0594

错误消息

浮点常数超出“type”类型的范围

给浮点型变量所赋的值对于此数据类型的变量而言太长。有关数据类型允许的值范围的信息，请参见[整型表](#)。

下面的示例生成 CS0594:

```
// CS0594.cs
namespace MyNamespace
{
    public class MyClass
    {
        public static void Main()
        {
            float f = 6.77777777777E400;    // CS0594, value too large
        }
    }
}
```

编译器错误 CS0596

错误消息

Guid 属性必须用 ComImport 属性指定

在使用 **ComImport** 属性时必须有 **Guid** 属性。

下面的示例生成 CS0596:

```
// CS0596.cs
using System.Runtime.InteropServices;

namespace x
{
    [ComImport] // CS0596
    // try the following line to resolve this CS0596
    // [ComImport, Guid("00000000-0000-0000-0000-000000000001")]
    public class a
    {
    }

    public class b
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS0599

错误消息

命名属性参数“argument”的值无效

传递给属性的一个参数无效。

编译器错误 CS0601

错误消息

必须在标记为“static”和“extern”的方法上指定 DllImport 属性

DllImport 属性用在了不具有正确访问关键字的方法中。

下面的示例生成 CS0601:

```
// CS0601.cs
using System.Runtime.InteropServices;
using System.Text;

public class C
{
    [DllImport("KERNEL32.DLL")]
    extern int GetCurDirectory(int bufSize, StringBuilder buf); // CS0601
    // Try the following line instead:
    // static extern int GetCurDirectory(int bufSize, StringBuilder buf);
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

编译器错误 CS0609

错误消息

不能在标记为 override 的索引器上设置 IndexerName 属性

名称特性 (**IndexerNameAttribute**) 不能应用于本身为重写的索引属性。有关更多信息, 请参见[索引器](#)。

下面的示例生成 CS0609:

```
// CS0609.cs
using System;
using System.Runtime.CompilerServices;

public class idx
{
    public virtual int this[int iPropIndex]
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }
}

public class MonthDays : idx
{
    [IndexerName("MonthInfoIndexer")] // CS0609, delete to resolve this CS0609
    public override int this[int iPropIndex]
    {
        get
        {
            return 0;
        }
        set
        {
        }
    }
}

public class test
{
    public static void Main( string[] args )
    {
    }
}
```

编译器错误 CS0610

错误消息

字段或属性不能是“type”类型

有些类型不能用作字段或属性。这些类型包括 **System.ArgIterator** 和 **System.TypedReference**。

下面的示例由于将 **System.TypedReference** 用作字段而生成 CS0610:

```
// CS0610.cs
public class MainClass
{
    System.TypedReference i; // CS0610
    public static void Main ()
    {
    }

    public static void Test(System.TypedReference i) // OK
    {
    }
}
```

编译器错误 CS0611

错误消息

数组元素不能是“type”类型

有些类型不能用作数组的类型。这些类型包括 **System.TypedReference** 和 **System.ArgIterator**。

下面的示例由于将 **System.TypedReference** 用作数据元素而生成 CS0611：

```
// CS0611.cs
public class a
{
    public static void Main()
    {
        System.TypedReference[] ao = new System.TypedReference [10]; // CS0611
        // try the following line instead
        // int[] ao = new int[10];
    }
}
```

编译器错误 CS0616

错误消息

"class"不是属性类

试图在属性块中使用非属性类。所有属性类型都必须从 [System.Attribute](#) 继承。

示例

下面的示例生成 CS0616。

```
// CS0616.cs
// compile with: /target:library
[CMYClass(i = 5)] // CS0616
public class CMYClass {}
```

下面的示例显示您可以如何定义属性：

```
// CreateAttrib.cs
// compile with: /target:library
using System;

[AttributeUsage(AttributeTargets.Class | AttributeTargets.Interface)]
public class MyAttr : Attribute
{
    public int Name = 0;
    public int Count = 0;

    public MyAttr (int iCount, int sName)
    {
        Count = iCount;
        Name = sName;
    }
}

[MyAttr(5, 50)]
class Class1 {}

[MyAttr(6, 60)]
interface Interface1 {}
```

编译器错误 CS0617

错误消息

"reference"不是有效的命名属性参数。命名属性(Attribute)参数必须是非只读、非静态或非常数的字段，或者是公共的和非静态的读写属性(Property)。

试图访问属性类的[私有成员](#)。

示例

下面的示例生成 CS0617。

```
// CS0617.cs
using System;

[AttributeUsage(AttributeTargets.Struct | 
                 AttributeTargets.Class | 
                 AttributeTargets.Interface)]
public class MyClass : Attribute
{
    public int Name;

    public MyClass (int sName)
    {
        Name = sName;
        Bad = -1;
        Bad2 = -1;
    }

    public readonly int Bad;
    public int Bad2;
}

[MyClass(5, Bad=0)] class Class1 {} // CS0617
[MyClass(5, Bad2=0)] class Class2 {}
```

编译器错误 CS0619

错误消息

"member"已过时 :"text"

类成员是用 [Obsolete](#) 属性标记的, 以致在引用该类成员时会发出错误。

下面的示例生成 CS0619:

```
// CS0619.cs
using System;

public class C
{
    [Obsolete("Use newMethod instead", true)] // generates an error on use
    public static void m()
    {
    }

    // this is the method you should be using
    public static void newMethod()
    {
    }
}

class MyClass
{
    public static void Main()
    {
        C.m(); // CS0619
    }
}
```

编译器错误 CS0620

错误消息

索引器不能有 void 类型

索引器的返回类型不能是 **void**。索引器必须返回某个值。

下面的示例生成 CS0620:

```
// CS0620.cs
class MyClass
{
    public static void Main()
    {
        MyClass test = new MyClass();
        System.Console.WriteLine(test[2]);
    }

    void this [int intI]    // CS0620, return type cannot be void
    {
        get
        {
            // will need to return some value
        }
    }
}
```

编译器错误 CS0621

错误消息

"member": 虚拟成员或抽象成员不能是私有的

不允许使用私有的 **virtual** 或 **abstract** 成员。

下面的示例生成 CS0621:

```
// CS0621.cs
abstract class MyClass
{
    private virtual void DoNothing1()    // CS0621
    {
    }

    private abstract void DoNothing2();   // CS0621

    public static void Main()
    {
    }
}
```

编译器错误 CS0622

错误消息

只能使用数组初始值设定项表达式为数组类型赋值。请尝试改用新的表达式。

在非数组的声明中使用了适合于初始化数组的语法。

示例

下面的示例生成 CS0622:

```
// CS0622.cs
using System;

public class Test
{
    public static void Main ()
    {
        Test t = { new Test() };    // CS0622
        // Try the following instead:
        // Test[] t = { new Test() };
    }
}
```

编译器错误 CS0623

错误消息

数组

数组未正确初始化。

编译器错误 CS0625

错误消息

"field": 标记为 StructLayout(LayoutKind.Explicit) 的实例字段类型必须具有 FieldOffset 属性

当结构标记为显式 **StructLayout** 属性时, 结构中的所有字段必须具有 **FieldOffset** 属性。有关更多信息, 请参见 [StructLayoutAttribute 类](#)。

下面的示例生成 CS0625:

```
// CS0625.cs
// compile with: /target:library
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
struct A
{
    public int i;    // CS0625 not static; an instance field
}

// OK
[StructLayout(LayoutKind.Explicit)]
struct B
{
    [FieldOffset(5)]
    public int i;
}
```

编译器错误 CS0629

错误消息

条件成员“member”无法在类型“Type Name”中实现接口成员“base class member”

[Conditional](#) 属性不能用在接口的实现中。

下面的示例生成 CS0629:

```
// CS0629.cs
interface MyInterface
{
    void MyMethod();
}

public class MyClass : MyInterface
{
    [System.Diagnostics.Conditional("debug")]
    public void MyMethod()      // CS0629, remove the Conditional attribute
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0631

错误消息

ref 和 out 参数在此上下文中无效

indexer 的声明不能包括对 ref 或 out 参数的使用。

示例

下面的示例生成 CS0631:

```
// CS0631.cs
public class MyClass
{
    public int this[ref int i]    // CS0631
    // try the following line instead
    // public int this[int i]
    {
        get
        {
            return 0;
        }
    }
}

public class MyClass2
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0633

错误消息

"attribute"属性的参数必须是有效的标识符

传递给 [ConditionalAttribute](#) 或 [IndexerNameAttribute](#) 属性的任何参数都必须是有效的标识符。这意味着它不能包含出现在标识符中时非法的字符，如“+”。

示例

此示例使用 [ConditionalAttribute](#) 阐释了 CS0633。下面的示例生成 CS0633。

```
// CS0633a.cs
#define DEBUG
using System.Diagnostics;
public class Test
{
    [Conditional("DEB+UG")]    // CS0633
    // try the following line instead
    // [Conditional("DEBUG")]
    public static void Main() { }
}
```

此示例使用 [IndexerNameAttribute](#) 阐释了 CS0633。

```
// CS0633b.cs
// compile with: /target:module
#define DEBUG
using System.Runtime.CompilerServices;
public class Test
{
    [IndexerName("Invalid+Identifier")]    // CS0633
    // try the following line instead
    // [IndexerName("DEBUG")]
    public int this[int i]
    {
        get { return i; }
    }
}
```

编译器错误 CS0634

错误消息

"arg":参数只对 System.Interop.UnmanagedType.CustomMarshaller 类型的封送有效

传递给 **Marshal** 属性的参数之一只能在封送格式为 **System.Interop.UnmanagedType.CustomMarshaller** 时使用。

编译器错误 CS0635

错误消息

"attribute": System.Interop.UnmanagedType.CustomMarshaller 需要命名参数 ComType 和 Marshal

ComType 和 **Marshal** 参数必须在封送格式为 **System.Interop.UnmanagedType.CustomMarshaller** 时指定。

编译器错误 CS0636

错误消息

FieldOffset 属性只能放置在标记为 StructLayout(LayoutKind.Explicit) 的类型的成员上

如果结构包含标记为 **FieldOffset** 属性的任何成员，必须在该属性的声明上使用 **StructLayout(LayoutKind.Explicit)** 属性。有关更多信息，请参见 [FieldOffset](#)。

下面的示例生成 CS0636：

```
// CS0636.cs
using System;
using System.Runtime.InteropServices;

// To resolve the error, uncomment the following line:
// [StructLayout(LayoutKind.Explicit)]
struct Worksheet
{
    [FieldOffset(4)]public int i;    // CS0636
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

编译器错误 CS0637

错误消息

静态或常数字段上不允许 FieldOffset 属性

FieldOffset 属性不能用在使用 static 或 const 标记的字段上。

下面的示例生成 CS0637:

```
// CS0637.cs
using System;
using System.Runtime.InteropServices;

[StructLayout(LayoutKind.Explicit)]
public class MainClass
{
    [FieldOffset(3)] // CS0637
    public static int i;
    public static void Main ()
    {
    }
}
```

编译器错误 CS0641

错误消息

"attribute": 属性只在由 System.Attribute 派生的类中有效

使用的属性只能用在从 **System.Attribute** 派生的类上。

下面的示例生成 CS0641:

```
// CS0641.cs
using System;

[AttributeUsage(AttributeTargets.All)]
public class NonAttrClass // CS0641
// try the following line instead
// public class NonAttrClass : Attribute
{
}

class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0643

错误消息

"arg"重复命名属性参数

参数 *arg* 在用户定义的属性上被指定了两次。有关更多信息, 请参见[属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0643:

```
// CS0643.cs
using System;
using System.Runtime.InteropServices;

[AttributeUsage(AttributeTargets.Class)]
public class MyAttribute : Attribute
{
    public MyAttribute()
    {
    }

    public int x;
}

[MyAttribute(x = 5, x = 6)] // CS0643, error setting x twice
// try the following line instead
// [MyAttribute(x = 5)]
class MyClass
{
}

public class MainClass
{
    public static void Main ()
    {
    }
}
```

编译器错误 CS0644

错误消息

"class1"不能从特殊类"class2"继承

类不能从下列任何基类中显式继承:

- **System.Enum**
- **System.ValueType**
- **System.Delegate**
- **System.Array**

这些基类被编译器用作隐式基类。例如, **System.ValueType** 是结构的隐式基类。

下面的示例生成 CS0644:

```
// CS0644.cs
class MyClass : System.ValueType // CS0644
{ }
```

编译器错误 CS0645

错误消息

标识符太长

类名称或其他标识符的长度不能超过 512 个字符。

编译器错误 CS0646

错误消息

不能对包含索引器的类型指定 DefaultMember 属性

如果类或其他类型指定了 **System.Reflection.DefaultMemberAttribute**, 则不能包含索引器。有关更多信息, 请参见[属性](#)。

下面的示例生成 CS0646:

```
// CS0646.cs
// compile with: /target:library
[System.Reflection.DefaultMemberAttribute("x")]    // CS0646
class MyClass
{
    public int this[int index]    // an indexer
    {
        get
        {
            return 0;
        }
    }

    public int x = 0;
}

// OK
[System.Reflection.DefaultMemberAttribute("x")]
class MyClass2
{
    public int prop
    {
        get
        {
            return 0;
        }
    }

    public int x = 0;
}

class MyClass3
{
    public int this[int index]    // an indexer
    {
        get
        {
            return 0;
        }
    }

    public int x = 0;
}
```

编译器错误 CS0647

错误消息

发出“attribute”属性时出错 --“reason”

下面的示例生成 CS0647:

```
// CS0647.cs
using System.Runtime.InteropServices;

[Guid("z")]    // CS0647, incorrect uuid format.
// try the following line instead
// [Guid("00000000-0000-0000-0000-000000000001")]
public class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0648

错误消息

"type"是该语言不支持的类型

从其他语言(可能为 C++)生成的元数据包含了未标记为托管的类型。有关该类型的信息包含在元数据中, 但该类型不能用于用 C# 编写的程序。

编译器错误 CS0650

错误消息

语法错误，错误的数组声明符。若要声明托管数组，秩说明符应位于变量的标识符之前。若要声明固定大小缓冲区字段，应在字段类型之前使用 `fixed` 关键字。

数组未正确声明。请注意，固定大小缓冲区的语法与数组的不同。

示例

下面的示例生成 CS0650。

```
// CS0650.cs
public class MyClass
{
    public static void Main()
    {
        int myarray[2];    // CS0650

        // OK
        int[] myarray2 = new int[2] {1,2};
        myarray2[0] = 0;
    }
}
```

编译器错误 CS0653

错误消息

无法应用属性类“class”，因为它是抽象的

`abstract` 自定义属性类不能用作属性。

下面的示例生成 CS0653：

```
// CS0653.cs
using System;

public abstract class MyAttribute : Attribute
{
}

public class My2Attribute : MyAttribute
{
}

[My] // CS0653
// try the following line instead
// [My2]
class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0655

错误消息

"parameter"不是有效的命名属性参数, 因为它不是有效的属性参数类型

有关属性的有效参数类型的讨论, 请参见[属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0655:

```
// CS0655.cs
using System;

class MyAttribute : Attribute
{
    // decimal is not valid attribute parameter type
    public decimal d = 0;
    public int e = 0;
}

[My(d = 0)]    // CS0655
// Try the following line instead:
// [My(e = 0)]
class C
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0656

错误消息

缺少编译器要求的成员“object.member”

存在以下问题之一：

- 公共语言运行库安装损坏。
- 引用的程序集定义了也可在公共语言运行库中找到的类型。但程序集的类型没有按照 C# 编译器需要的方式进行定义。

检查引用以确保正使用正确版本的公共语言运行库。

编译器警告(等级 1)CS0657

错误消息

"attribute modifier"对于此声明不是有效的属性位置。此声明的有效属性位置为"location"

编译器在无效的位置找到了属性修饰符。有关更多信息, 请参见[属性目标](#)。

下面的示例生成 CS0657:

```
// CS0657.cs
// compile with: /target:library
public class TestAttribute : System.Attribute {}
[return: Test]    // CS0657 return not valid on a class
class Class1 {}
```

编译器错误 CS0662

错误消息

"method"不能在 ref 参数上仅指定 Out 属性。请同时使用 In 和 Out 属性, 或者两者都不使用。

接口方法的一个参数使用了只有 Out 属性的 ref。使用 Out 属性的 ref 参数还必须使用 In 属性。

下面的示例生成 CS0662:

```
// CS0662.cs
using System.Runtime.InteropServices;

interface I
{
    void method([Out] ref int i);    // CS0662
    // try one of the following lines instead
    // void method(ref int i);
    // void method([Out, In]ref int i);
}

class test
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0663

错误消息

不能定义仅在 `ref` 和 `out` 上有差别的重载方法。

不允许使用仅在参数的 `ref` 和 `out` 用法上有差异的方法。

下面的示例生成 CS0663:

```
// CS0663.cs
class TestClass
{
    public static void Main()
    {

        public void Test(ref int i)
        {

        }

        public void Test(out int i) // CS0663
        {
        }
    }
}
```

编译器错误 CS0664

错误消息

不能隐式地将 Double 类型转换为“type”类型; 请使用“suffix”后缀创建此类型

赋值无法完成, 请使用后缀更正指令。每个类型的文档确定该类型的相应后缀。

下面的示例生成 CS0664:

```
// CS0664.cs
class M
{
    static void Main()
    {
        decimal m = 1.0;    // CS0664
        // try the following line instead
        // decimal m = 1.0M;
        System.Console.WriteLine(m);
    }
}
```

编译器错误 CS0666

错误消息

"member": 结构中已声明新的保护成员

`struct` 不能为 **abstract**, 而应始终为隐式 **sealed**。有关更多信息, 请参见[继承\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0666:

```
// CS0666.cs
class M
{
    static void Main()
    {
    }
}

struct S
{
    protected int x;    // CS0666
}
```

编译器错误 CS0667

错误消息

"无效功能"这一功能已不推荐使用。请改用"有效功能"。

您所尝试使用的功能已不推荐使用。请更新代码以改用有效功能。

编译器错误 CS0668

错误消息

两个索引器的名称不同;类型中的每个索引器上都必须用相同的名称使用 `IndexerName` 属性

对于类型中的所有索引器, 传递给 `IndexerName` 属性的值必须是相同的。有关 `IndexerName` 属性的更多信息, 请参见 [IndexerNameAttribute 类](#)。

下面的示例生成 CS0668:

```
// CS0668.cs
using System;
using System.Runtime.CompilerServices;

class IndexerClass
{
    [IndexerName("IName1")]
    public int this [int index]    // indexer declaration
    {
        get
        {
            return index;
        }
        set
        {
        }
    }

    [IndexerName("IName2")]
    public int this [string s]     // CS0668, change IName2 to IName1
    {
        get
        {
            return int.Parse(s);
        }
        set
        {
        }
    }

    void Main()
    {
    }
}
```

编译器错误 CS0669

错误消息

具有 ComImport 属性的类不能有用户定义的构造函数

公共语言运行库中的 COM interop 层为 [ComImport](#) 类提供构造函数。因此，COM 对象可以在运行库中作为托管对象使用。

下面的示例生成 CS0669：

```
// CS0669.cs
using System.Runtime.InteropServices;
[ComImport, Guid("00000000-0000-0000-0000-000000000001")]
class TestClass
{
    TestClass() // CS0669, delete constructor to resolve
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0670

错误消息

字段不能有 void 类型

字段被声明成了 `void` 类型。

下面的示例生成 CS0670:

```
// CS0670.cs
class C
{
    void f;    // CS0670
    // try the following line instead
    // public int f;

    public static void Main()
    {
        C myc = new C();
        myc.f = 0;
    }
}
```

编译器错误 CS0673

错误消息

在 C# 中无法使用 System.Void - 使用 typeof(void) 获取 void 类型对象。

System.Void 不能用在 C# 中。

下面的示例生成 CS0673:

```
// CS0673.cs
class MyClass
{
    public static void Main()
    {
        System.Type t = typeof(System.Void);    // CS0673
        // try the following line instead
        // System.Type t = typeof(void);
    }
}
```

编译器错误 CS0674

错误消息

不要使用“System.ParamArrayAttribute”。而是使用“params”关键字。

C# 编译器不允许使用 [System.ParamArrayAttribute](#), 应使用 [params](#)。

下面的示例生成 CS0674:

```
// CS0674.cs
using System;
public class MyClass
{
    public static void UseParams([ParamArray] int[] list) // CS0674
        // try the following line instead
        // public static void UseParams(params int[] list)
    {
        for ( int i = 0 ; i < list.Length ; i++ )
            Console.WriteLine(list[i]);
        Console.WriteLine();
    }

    public static void Main()
    {
        UseParams(1, 2, 3);
    }
}
```

编译器错误 CS0677

错误消息

"variable": volatile 字段不能是"type"类型

用 **volatile** 关键字声明的字段必须是下列类型之一:

- 任何引用类型
- 任何指针类型(在 **unsafe** 的上下文中)
- **sbyte**、**byte**、**short**、**ushort**、**int**、**uint**、**char**、**float**、**bool** 等类型
- 基于上述任何一种类型的枚举类型

下面的示例生成 CS0677:

```
// CS0677.cs
class TestClass
{
    private volatile long i;    // CS0677

    public static void Main()
    {
    }
}
```

编译器错误 CS0678

错误消息

"variable":一个字段不能同时为 volatile 和 readonly

volatile 和 readonly 关键字的使用是互斥的。

下面的示例生成 CS0678:

```
// CS0678.cs
using System;

class TestClass
{
    private readonly volatile int i;    // CS0678
    // try the following line instead
    // private volatile int i;

    public static void Main()
    {
    }
}
```

编译器错误 CS0681

错误消息

修饰符“abstract”对字段无效。请尝试改用属性。

您不能使字段成为抽象的。不过，您可以有访问该字段的抽象属性。

示例

下面的示例生成 CS0681：

```
// CS0681.cs
// compile with: /target:library
abstract class C
{
    abstract int num; // CS0681
}
```

请尝试改用以下代码：

```
// CS0681b.cs
// compile with: /target:library
abstract class C
{
    public abstract int num
    {
        get;
        set;
    }
}
```

编译器错误 CS0682

错误消息

"type1"无法实现"type2", 因为该语言不支持它

当您试图实现用其他语言编写的接口, 而编译器不支持该接口时, 将发生此错误。

编译器错误 CS0683

错误消息

"explicitmethod" 显式方法实现无法实现"method", 因为它是一个访问器

下面的示例生成 CS0683:

```
// CS0683.cs
interface IExample
{
    int Test { get; }
}

class CExample : IExample
{
    int IExample.get_Test() { return 0; } // CS0683
    int IExample.Test { get{ return 0; } } // correct
}
```

编译器错误 CS0685

错误消息

条件成员“member”不能有 out 参数

在方法上使用 [ConditionalAttribute](#) 属性时，该方法不能有 out 参数。这是因为，如果方法调用编译为 nothing，则不会定义用于 out 参数的变量的值。若要避免此错误，请将 out 参数从条件方法声明中移除，或者不使用条件属性。

示例

下面的示例生成 CS0685：

```
// CS0685.cs
using System.Diagnostics;

class C
{
    [Conditional("DEBUG")]
    void trace(out int i) // CS0685
    {
        i = 1;
    }
}
```

编译器错误 CS0686

错误消息

访问符“accessor”无法实现类型“type”的接口成员“member”。使用显式接口实现。

提示:当实现一个接口时,如果其中包含的方法名称与属性或事件关联的自动生成的方法相冲突,会出现此错误。属性的 get/set 方法生成为 get_property 和 set_property,事件的 add/remove 方法生成为 add_event 和 remove_event。如果接口包含其中的任何一个方法,则发生冲突。要避免此错误,请使用显式接口实现来实现这些方法。要这样做,请将函数指定为:

```
Interface.get_property() { /* */ }
Interface.set_property() { /* */ }
```

示例

以下示例生成 CS0686:

```
// CS0686.cs
interface I
{
    int get_P();
}

class C : I
{
    public int P
    {
        get { return 1; } // CS0686
    }
}
// But the following is valid:
class D : I
{
    int I.get_P() { return 1; }
    public static void Main() {}
}
```

声明事件时也会发生此错误。事件结构自动生成 add_event 和 remove_event 方法,这可能和接口中的同名方法相冲突,如下示例所示:

```
// CS0686b.cs
using System;

interface I
{
    void add_OnMyEvent(EventHandler e);
}

class C : I
{
    public event EventHandler OnMyEvent
    {
        add { } // CS0686
        remove { }
    }
}

// Correct (using explicit interface implementation):
class D : I
{
    void I.add_OnMyEvent(EventHandler e) {}
    public static void Main() {}
}
```


编译器错误 CS0687

错误消息

命名空间别名限定符“::”始终解析为类型或命名空间，因此在这里是非法的。请考虑改用“.”。

如果您使用的某些内容在意外的位置被分析器解释为类型，将出现此错误。类型或命名空间名称只有在成员访问表达式中使用成员访问（.）运算符时才有效。如果您在另一个上下文中使用了全局范围运算符（::），可能会出现此错误。

示例

下面的示例生成 CS0687：

```
// CS0687.cs

using M = Test;
using System;

public class Test
{
    public static int x = 77;

    public static void Main()
    {
        Console.WriteLine(M::x); // CS0687
        // To resolve use the following line instead:
        // Console.WriteLine(M.x);
    }
}
```

编译器错误 CS0689

错误消息

不能从“identifier”派生，因为它是类型参数

泛型类的基类或接口不能由类型参数指定。请从一个特定的类或接口派生，或者从一个特定的泛型类派生，或者包含一个未知的类型作为成员。

下面的示例生成 CS0689：

```
// CS0689.cs
class A<T> : T    // CS0689
{}
```

编译器错误 CS0690

错误消息

输入文件“file”包含无效的元数据。

可以打开元数据文件，但由于某种本地化的问题，元数据已损坏。这与错误 [CS0009](#) 相似，只是此时您能够打开元数据文件。

PE (Process Executable) 是一个可执行文件。

编译器错误 CS0692

错误消息

重复的类型参数“identifier”

同一名称不能在类型参数列表中使用一次以上。请重命名或移除重复的类型参数。

示例

下面的示例生成 CS0692:

```
// CS0692.cs
// compile with: /target:library
class C <T, A, T>    // CS0692
{
}

class D <T, T>    // CS0692
{}
```

编译器错误 CS0694

错误消息

类型参数“identifier”与包含类型或方法同名

由于类型参数的名称不能与包含该类型参数的类型或方法同名，因此必须为该类型参数使用其他名称。

示例

下面的示例生成 CS0694。

```
// CS0694.cs
// compile with: /target:library
class C<C> {} // CS0694
```

除以上涉及泛型类的情况外，方法也可能发生此错误：

```
// CS0694_2.cs
// compile with: /target:library
class A
{
    public void F<F>(F arg); // CS0694
}
```

编译器错误 CS0695

错误消息

"generic type"无法同时实现"generic interface"和"generic interface", 因为它们对于某些类型参数替换可以统一

当一个泛型类实现同一泛型接口的多个参数化, 并且存在一个会使两个接口完全相同的类型参数替换时, 将出现此错误。若要避免此错误, 请仅实现一个接口, 或者更改类型参数以避免冲突。

下面的示例生成 CS0695:

```
// CS0695.cs
// compile with: /target:library

interface I<T>
{
}

class G<T1, T2> : I<T1>, I<T2> // CS0695
{}
```

编译器错误 CS0698

错误消息

"class"是一个属性类，无法从它派生泛型类型

从属性类派生的任何类都是属性。属性不能是泛型类型。

下面的示例生成 CS0698：

```
// CS0698.cs
class C<T> : System.Attribute // CS0698
{
}
```

编译器错误 CS0699

错误消息

"generic"未定义类型参数"identifier"

泛型定义中使用了一个类型参数，但在该泛型的类型参数的声明列表中未找到该类型参数。如果用于该类型参数的名称不一致，则会发生这种情况。

下面的示例生成 CS0699：

```
// CS0699.cs
class C<T> where U : I    // CS0699 - U is not a valid type parameter
{
}
```

编译器错误 CS0701

错误消息

"identifier"不是有效的约束。作为约束使用的类型必须是接口、非密封类或类型参数。

如果将密封类型用作约束，则会发生此错误。若要解决此错误，请仅将非密封类型用作约束。

示例

下面的示例生成 CS0701。

```
// CS0701.cs
// compile with: /target:library
class C<T> where T : System.String {}    // CS0701
class D<T> where T : System.Attribute {}   // OK
```

编译器错误 CS0702

错误消息

约束不能是特殊类“identifier”

下面的类型不能用作约束: System.Array、 System.Delegate、 System.Enum 和 System.ValueType。

示例

下面的示例生成 CS0702:

```
// CS0702.cs
class C<T> where T : System.Array // CS0702
{ }
```

编译器错误 CS0703

错误消息

可访问性不一致: 约束类型“identifier”的可访问性比“identifier”低。

一个约束不能强制泛型参数比泛型类本身的可访问性低。在下面的示例中，虽然泛型类 `C<T>` 被声明为 `public`, 但约束尝试强制 `T` 实现内部接口。即使允许这样做，也只有具有内部访问的客户端才能为该类创建参数，因此实际上只有具有内部访问的客户端才能使用该类。

为消除此错误，请确保泛型类的访问级别的限制性不低于界限中出现的任何类或接口。

下面的示例生成 CS0703:

```
// CS0703.cs
internal interface I {}
public class C<T> where T : I // CS0703 - I is internal; C<T> is public
{
}
```

编译器错误 CS0704

错误消息

"type"是一个类型参数, 无法在其中执行成员查找

无法通过类型参数指定内部类型。请尝试显式使用所需的类型。

示例

下面的示例生成 CS0704:

```
// CS0704.cs
class B
{
    public class I { }

    class C<T> where T : B
    {
        T.I f; // CS0704 - member lookup on type parameter
        // Try this instead:
        // B.I f;
    }

    class CMain
    {
        public static void Main() {}
    }
}
```

编译器错误 CS0706

错误消息

无效的约束类型。作为约束使用的类型必须是接口、非密封类或类型参数。

当在约束子句中使用无效的构造时，将出现此错误。若要避免此错误，请使用接口或非密封类取代导致此错误的构造。

示例

下面的示例生成 CS0706。

```
// CS0706.cs
// compile with: /target:library
class A {}
class C<T> where T : int[] {} // CS0706
class D<T> where T : A {} // OK
```

编译器错误 CS0708

错误消息

"field": 不能在静态类中声明实例成员

如果在声明为静态的类中声明非静态成员，则会发生此错误。不可能创建静态类的实例，因此实例变量没有意义。**static** 关键字应该应用于静态类的所有成员。

下面的示例生成 CS0708：

```
// CS0708.cs
// compile with: /target:library
public static class C
{
    int i; // CS0708
    static int j; // OK
}
```

编译器错误 CS0709

错误消息

"derived class": 无法从静态类"base class"派生

无法实例化静态类或从其派生。即，静态类不能作为任何其他类的基类。

示例

下面的示例生成 CS0709。

```
// CS0709.cs
// compile with: /target:library
public static class Base {}
public class Derived : Base {}    // CS0709
```

编译器错误 CS0710

错误消息

静态类不能有实例构造函数

静态类无法实例化，因此它不需要构造函数。若要避免此错误，请从静态类中移除所有构造函数，或者如果您确实希望构造实例，请将类设为非静态类。

下面的示例生成 CS0710：

```
// CS0710.cs
public static class C
{
    public C() // CS0710
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0711

错误消息

静态类不能包含析构函数

静态类不能实例化，因为它不需要构造函数或析构函数。若要避免此错误，请从静态类中移除任何析构函数，或者，如果您确实希望构造并销毁实例，请使类成为非静态。

下面的示例生成 CS0711：

```
// CS0711.cs
public static class C
{
    ~C() // CS0711
    {

    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0712

错误消息

无法创建静态类“static class”的实例

不可能创建静态类的实例。静态类被设计为包含静态字段和方法，但不能实例化。

示例

下面的示例生成 CS0712:

```
// CS0712.cs
public static class SC
{
}

public class CMain
{
    public static void Main()
    {
        SC sc = new SC(); // CS0712
    }
}
```

编译器错误 CS0713

错误消息

静态类“static type”不能从类型“type”派生。静态类必须从对象派生。

如果允许这样做，静态类将从基类继承方法和非静态成员，这样它就不是静态的了。因此不允许这样做。

下面的示例生成 CS0713：

```
// CS0713.cs
public class Base
{
}

public static class Derived : Base // CS0713
{
}

public class CMain
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0714

错误消息

"static type": 静态类不能实现接口

接口可能在对象上定义非静态方法，因此不能由静态类实现。若要解决此错误，请确保您的类没有试图实现任何接口。

示例

下面的示例生成 CS0714:

```
// CS0714.cs
interface I
{
}

public static class C : I // CS0714
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0715

错误消息

"static class": 静态类不能包含用户定义的运算符

用户定义的运算符作用于类的实例。静态类无法实例化，所以不可能创建运算符作用于的实例。因此，不允许对静态类使用用户定义的运算符。

下面的示例生成 CS0715：

```
// CS0715.cs
public static class C
{
    public static C operator+(C c) // CS0715
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0716

错误消息

无法转换为静态类型“type”

如果您的代码使用强制转换来转换为静态类型，则会发生此错误。由于一个对象不可能成为静态类型的实例，因此强制转换为静态类型永远没有意义。

示例

下面的示例生成 CS0716：

```
// CS0716.cs

public static class SC
{
    static void F() { }

public class Test
{
    public static void Main()
    {
        object o = new object();
        System.Console.WriteLine((SC)o); // CS0716
    }
}
```

编译器错误 CS0717

错误消息

"static class": 静态类不能用作约束

静态类仅包含静态成员而不包含实例成员，因此不能扩展。由于静态类不能扩展，这使它们作为类型参数和约束时没有意义，因为不存在可以实现静态类专用化的类型。

示例

下面的示例生成 CS0717：

```
// CS0717.cs

public static class SC
{
    public static void F()
    {
    }
}

public class G<T> where T : SC // CS0717
{
    public static void Main()
    {
    }
}
```

编译器错误 CS0718

错误消息

"type": 静态类型不能用作泛型参数

静态类型无法实例化，因此不能用作泛型参数。若要解决此错误，请从泛型参数中移除静态类型。

示例

下面的示例生成 CS0718：

```
// CS0718.cs
public static class SC
{
    public static void F()
    {
    }
}

public class G<T>
{
}

public class CMain
{
    public static void Main()
    {
        G<SC> gsc = new G<SC>(); // CS0718
    }
}
```

编译器错误 CS0719

错误消息

"type": 数组元素不能是静态类型

静态类型的数组没有意义，因为数组元素是实例，而不可能创建静态类型的实例。

下面的示例生成 CS0719：

```
// CS0719.cs
public static class SC
{
    public static void F()
    {
    }
}

public class CMain
{
    public static void Main()
    {
        SC[] sca = new SC[10]; // CS0719
    }
}
```

编译器错误 CS0720

错误消息

"static class": 不能在静态类中声明索引器

索引器在静态类中没有意义，因为它们只能和实例一起使用，而且不可能创建静态类型的实例。

示例

下面的示例生成 CS0720：

```
// CS0720.cs

public static class Test
{
    public int this[int index] // CS0720
    {
        get { return 1; }
        set {}
    }

    static void Main() {}
}
```

编译器错误 CS0721

错误消息

"type": 静态类型不能用作参数

静态类型作为参数没有意义。由于不能创建静态类型的实例，所以没有实例可以作为参数传递。

下面的示例生成 CS0721：

```
// CS0721.cs
public static class SC
{
}

public class CMain
{
    public void F(SC sc) // CS0721
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0722

错误消息

"type": 静态类型不能用作返回类型

由于无法创建静态类型的实例，所以将静态类型用作返回类型没有意义。

下面的示例生成 CS0722:

```
// CS0722.cs
public static class SC
{
}

public class CMain
{
    public SC F() // CS0722
    {
        return null;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS0723

错误消息

无法声明静态类型“type”的变量

无法创建静态类型的实例。

下面的示例生成 CS0723:

```
// CS0723.cs
public static class SC
{
}

public class CMain
{
    public static void Main()
    {
        SC sc = null; // CS0723
    }
}
```

编译器错误 CS0724

错误消息

在嵌套到最里层的 catch 子句内部的 finally 子句内不允许使用不带参数的 throw 语句

下面的示例生成 CS0724, 原因是 **throw** 语句在 **finally** 子句块内。

示例

下面的示例生成 CS0724。

```
// CS0724.cs
using System;

class X
{
    static void Test()
    {
        try
        {
            throw new Exception();
        }
        catch
        {
            try
            {
            }
            finally
            {
                throw; // CS0724
            }
        }
    }

    static void Main()
    {
    }
}
```

编译器错误 CS0726

错误消息

"format specifier"不是有效的格式说明符

在调试器中发生此错误。当您向某个调试器窗口键入变量名称时，您可以在其后跟一个逗号，然后再跟一个格式说明符。例如:myInt, h 或 myString,nq。当编译器不识别 [C# 中的格式说明符](#)时，会发生此错误。

编译器错误 CS0727

错误消息

无效的格式说明符

在调试器中发生此错误。当您向某个调试器窗口键入变量名称时，您可以在其后跟一个逗号，然后再跟一个格式说明符。例如：myInt, h; 或 myString,nq。当编译器完全无法分析您所键入的内容时会出现此错误。要解决此错误，请重新键入变量名称，您可以选择在其后跟一个逗号和[有效的格式说明符](#)。

编译器错误 CS0729

错误消息

类型“type”在此程序集中定义，但为其指定了类型转发器。

不能对在同一程序集中定义的类型使用类型转发器。

示例

下面的示例生成 CS0729。

```
// CS0729.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(TestClass))]    // CS0729
class TestClass {}
```

编译器错误 CS0730

错误消息

由于类型“type”是“type”的嵌套类型，因此无法转发该类型

尝试转发嵌套类时会生成此错误。

示例

下面的示例生成 CS0730。它由两个源文件组成。首先，编译库文件 CS0730a.cs，然后编译引用该库文件的文件 CS0730.cs。

```
// CS0730a.cs
// compile with: /t:library
public class Outer
{
    public class Nested {}
}
```

```
// CS0730.cs
// compile with: /t:library /r:CS0730a.dll
using System.Runtime.CompilerServices;

[assembly:TypeForwardedToAttribute(typeof(Outer.Nested))]    // CS0730
[assembly:TypeForwardedToAttribute(typeof(Outer))]    // OK
```

编译器错误 CS0731

错误消息

程序集“assembly”中类型“type”的类型转发器导致循环

仅当导入的源数据格式错误时才会发生此错误。仅使用 C# 源代码不会出现此错误。

示例

下面的示例生成 CS0731。此示例由三个文件组成：

1. Circular.il
2. Circular2.il
3. CS0731.cs

首先将 .IL 文件编译为库，然后编译引用这两个文件的 .cs 代码。

```
// Circular.il
// compile with: /DLL /out=Circular.dll
.assembly extern circular2
{
    .ver 0:0:0:0
}
.assembly extern circular3
{
    .ver 0:0:0:0
}
.assembly extern mscorel
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89) // .z\V.4..
    .ver 2:0:0:0
}
.assembly Circular
{
    .custom instance void [mscorel]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::ctor(int32) = ( 01 00 08 00 00 00 00 00 )
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
    .assembly extern circular2
}
.module Circular.dll
// MVID: {880C2329-C915-42A0-83E9-9D10C3E6DBD0}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003      // WINDOWS_CUI
.corflags 0x00000001   // ILOONLY
// Image base: 0x04E40000
// ===== CLASS MEMBERS DECLARATION =====
.class public abstract auto ansi sealed beforefieldinit User
    extends [mscorel]System.Object
{
    .method public hidebysig static class [circular2]Circular.Referenced.TypeForwarder
        F() cil managed
    {
        .maxstack 1
        newobj    instance void [circular2]Circular.Referenced.TypeForwarder::ctor()
        ret
    }
}
```

```
// Circular2.il
// compile with: /DLL /out=Circular2.dll
.assembly extern mscorel
{
    .publickeytoken = (B7 7A 5C 56 19 34 E0 89 )    // .z\V.4..
    .ver 2:0:0:0
}
.assembly extern Circular
{
    .ver 0:0:0:0
}
.assembly circular2
{
    .custom instance void [mscorel]System.Runtime.CompilerServices.CompilationRelaxationsAttribute::ctor(int32) = ( 01 00 08 00 00 00 00 00 )
    .hash algorithm 0x00008004
    .ver 0:0:0:0
}
.class extern forwarder Circular.Referenced.TypeForwarder
{
    .assembly extern Circular
}
.module circular2.dll
// MVID: {8B3BE5C8-DBE1-49C4-BC72-DF35F0387C21}
.imagebase 0x00400000
.file alignment 0x00000200
.stackreserve 0x00100000
.subsystem 0x0003      // WINDOWS_CUI
.corflags 0x00000001   // ILOONLY
// Image base: 0x04E40000
```

```
// CS0731.cs
// compile with: /reference:circular.dll /reference:circular2.dll
// CS0731 expected
class A {
    public static void Main() {
        User.F();
    }
}
```

编译器错误 CS0733

错误消息

无法转发泛型类型“GenericType<>”

示例

下面的示例生成 CS0733。将第一个文件编译为库，然后在编译第二个文件时引用它。

```
// CS0733a.cs
// compile with: /target:library
public class GenericType<T>
{
}
```

```
// CS0733.cs
// compile with: /target:library /r:CS0733a.dll
[assembly: System.Runtime.CompilerServices.TypeForwardedTo(typeof(GenericType<int>))] // CS0733
```

编译器错误 CS0734

错误消息

仅在生成“module”的目标类型时才可以指定 /moduleassemblyname 选项

仅在生成 .netmodule 时才可以使用编译器选项 **/moduleassemblyname**。有关更多信息, 请参见 [/moduleassemblyname\(指定模块的友元程序集\)\(C# 编译器选项\)](#)。

有关生成 .netmodule 的更多信息, 请参见 [/target:module\(创建模块以添加到程序集\)\(C# 编译器选项\)](#)。

示例

下面的示例生成 CS0734。若要解决此问题, 请将 **/target:module** 添加到编译中。

```
// CS0734.cs
// compile with: /moduleassemblyname:A
// CS0734 expected
public class Test {}
```

编译器错误 CS1001

错误消息

应输入标识符

未提供标识符。例如，当声明枚举时，必须指定成员。

下面的示例生成 CS1001：

```
// CS1001.cs
public class clx
{
    enum splitch : int
    {
        'a', 'b' // CS1001, 'a' is not a valid int identifier
    };

    public static void Main()
    {
    }
}
```

即使编译器不使用参数名(例如，在接口定义中)，也需要参数名。需要这些参数，以便使用接口的程序员能够对参数的含义有所了解。

```
// CS1001-2.cs
// compile with: /target:library
interface IMyTest
{
    void TestFunc1(int, int); // CS1001
}

class CMyTest : IMyTest
{
    void IMyTest.TestFunc1(int a, int b)
    {
    }
}
```

编译器错误 CS1002

错误消息

应输入 ;

编译器检测到缺少分号。

下面的示例生成 CS1002:

```
// CS1002.cs
namespace x
{
    abstract public class clx
    {
        int i    // CS1002, missing semicolon

        public static int Main()
        {
            return 0;
        }
    }
}
```

编译器错误 CS1003

错误消息

语法错误, 应输入 "char"

编译器将因若干错误条件中的任何一个而生成该错误。请查看您的代码以找到语法错误。

下面的示例生成 CS1003:

```
// CS1003.cs
public class b
{
    public static void Main()
    {
        int[] a;
        a[]; // CS1003
    }
}
```

编译器错误 CS1004

错误消息

重复的“modifier”修饰符

检测到重复的修饰符, 如 **access** 修饰符。

下面的示例生成 CS1004:

```
// CS1004.cs
public class clx
{
    public public static void Main() // CS1004, two public keywords
    {
    }
}
```

编译器错误 CS1007

错误消息

属性访问器已经定义

当声明属性时, 还必须声明它的访问器方法。不过, 一个属性不能有多个 **get** 访问器方法或多个 **set** 访问器方法。

示例

下面的示例生成 CS1007:

```
// CS1007.cs
public class clx
{
    public int MyProperty
    {
        get
        {
            return 0;
        }
        get // CS1007, this is the second get method
        {
            return 0;
        }
    }

    public static void Main() {}
}
```

编译器错误 CS1008

错误消息

应输入类型 byte、sbyte、short、ushort、int、uint、long 或 ulong

某些数据类型(如 [enums](#))只能声明为保存指定类型的数据。

下面的示例生成 CS1008:

```
// CS1008.cs
abstract public class clk
{
    enum splitch : char // CS1008, char not valid type for enums
    {
        x, y, z
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1009

错误消息

无法识别的转义序列

在 `string` 中反斜杠 (\) 的后面是一个意外的字符。编译器需要一个有效的转义符;有关更多信息, 请参见[字符转义](#)。

下面的示例生成 CS1009:

```
// CS1009-a.cs
class MyClass
{
    static void Main()
    {
        string a = "\m";    // CS1009
        // try the following line instead
        // string a = "\t";
    }
}
```

发生该错误的原因通常是在文件名中使用了反斜杠字符, 例如:

```
string filename = "c:\myFolder\myFile.txt";
```

若要纠正该错误, 请使用“\\”或前面带有 @ 且用引号括起的字符串, 如下面的示例所示:

```
// CS1009-b.cs
class MyClass
{
    static void Main()
    {
        string filename = "c:\myFolder\myFile.txt";    // CS1009
        // try the one of the following lines instead
        // string filename = "c:\\myFolder\\\\myFile.txt";
        // string filename = @"c:\\myFolder\\myFile.txt";
    }
}
```

编译器错误 CS1010

错误消息

常数中有换行符

一个[字符串](#)没有正确分隔。

下面的示例生成 CS1010:

```
// CS1010.cs
class Sample
{
    static void Main()
    {
        string a = "Hello World;    // CS1010, add end quote
    }
}
```

编译器错误 CS1011

错误消息

空字符

声明了 `char`, 并将其初始化为空。初始化 `char` 时必须指定字符。

下面的示例生成 CS1011:

```
// CS1011.cs
class Sample
{
    public char CharField = '';    // CS1011
}
```

编译器错误 CS1012

错误消息

字符串文本中字符太多

试图初始化具有多个字符的 `char` 常数。

执行数据绑定时也会发生 CS1012。例如，下面的代码行将发生错误：

```
<%# DataBinder.Eval(Container.DataItem, 'doctitle') %>
```

请尝试改用下面的代码行：

```
<%# DataBinder.Eval(Container.DataItem, "doctitle") %>
```

下面的示例生成 CS1012：

```
// CS1012.cs
class Sample
{
    static void Main()
    {
        char a = 'xx';    // CS1012
        char a2 = 'x';   // OK
        System.Console.WriteLine(a2);
    }
}
```

编译器错误 CS1013

错误消息

无效数字

编译器检测到不正确的数字。有关整型的更多信息, 请参见[整型表\(C# 参考\)](#)。

示例

下面的示例生成 CS1013:

```
// CS1013.cs
class Sample
{
    static void Main()
    {
        int i = 0x;    // CS1013
    }
}
```

编译器错误 CS1014

错误消息

应为 get 或 set 访问器

在属性声明中发现了方法声明。在属性中只能声明 **get** 和 **set** 方法。

有关属性的更多信息, 请参见[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1014。

```
// CS1014.cs
// compile with: /target:library
class Sample
{
    public int TestProperty
    {
        get
        {
            return 0;
        }
        int z; // CS1014  not get or set
    }
}
```

编译器错误 CS1015

错误消息

应输入对象、字符串或类类型

试图将预定义的数据类型传递到 `catch` 块。只有从 [System.Exception](#) 派生的数据类型才能传递到 `catch` 块中。有关异常的更多信息, 请参见 [异常处理语句\(C# 参考\)](#)。

示例

下面的示例生成 CS1015:

```
// CS1015.cs
class Sample
{
    static void Main()
    {
        try
        {
        }
        catch(int) // CS1015, int is not derived from System.Exception
        {
        }
    }
}
```

编译器错误 CS1016

错误消息

应输入命名属性参数

未命名的属性参数必须出现在命名参数之前。

示例

下面的示例生成 CS1016:

```
// CS1016.cs
using System;

[AttributeUsage(AttributeTargets.Class)]
public class HelpAttribute : Attribute
{
    public HelpAttribute(string url) // url is a positional parameter
    {
        m_url = url;
    }

    public string Topic = null; // Topic is a named parameter
    private string m_url = null;
}

[HelpAttribute(Topic="Samples", "http://intranet/inhouse")] // CS1016
// try the following line instead
//[HelpAttribute("http://intranet/inhouse", Topic="Samples")]
public class MainClass
{
    public static void Main ()
    {
    }
}
```

编译器错误 CS1017

错误消息

Try 语句已经有了空的 catch 块

一个不带任何参数的 **catch** 块必须是一系列 **catch** 块的最后一项。有关异常的更多信息, 请参见[异常处理语句\(C# 参考\)](#)。

示例

下面的示例生成 CS1017:

```
// CS1017.cs
using System;

namespace x
{
    public class b : Exception
    {
    }

    public class a
    {
        public static void Main()
        {
            try
            {

                catch // CS1017, must be last catch
                {

                }

                catch(b)
                {
                    throw;
                }
            }
        }
    }
}
```

编译器错误 CS1018

错误消息

应输入关键字“this”或“base”

编译器遇到了不完整的构造函数声明。

示例

下面的示例生成 CS1018 并建议了几种解决此错误的方法:

```
// CS1018.cs
public class C
{
}

public class a : C
{
    public a(int i)
    {
    }

    public a () : // CS1018
    // possible resolutions:
    // public a () resolves by removing the colon
    // public a () : base() calls C's default constructor
    // public a () : this(1) calls the assignment constructor of class a
    {
    }

    public static int Main()
    {
        return 1;
    }
}
```

编译器错误 CS1019

错误消息

应输入可重载的一元运算符

您具有返回其他类的值的一元运算符。如果希望进行该转换，则需要进行 **implicit** 或 **explicit** 转换。

下面的示例生成 CS1019：

```
// CS1019.cs
public class ii
{
    int i
    {
        get
        {
            return 0;
        }
    }
}

public class a
{
    public static a operator ii(a aa) // CS1019
    // try the following line instead
    //public static a operator ++(a aa)
    {
        return new a();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1020

错误消息

应输入可重载的二元运算符

试图定义[运算符重载](#), 但运算符不是二进制运算符(带有两个参数)。

下面的示例生成 CS1020:

```
// CS1020.cs
public class iii
{
    public static int operator ++(iii aa, int bb)    // CS1020, change ++ to +
    {
        return 0;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1021

错误消息

整数常数太大

赋为[整型](#)的值超出了无符号整数值的最大范围。

下面的示例生成 CS1021:

```
// CS1021.cs
enum F : int
{
    a=(int)2590000000000000000000000000 // CS1021
}

public class clx
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1022

错误消息

应输入类型、命名空间定义或文件尾

源代码文件中的大括号不成对。

下面的示例生成 CS1022:

```
// CS1022.cs
namespace x
{
}
} // CS1022
```

编译器错误 CS1023

错误消息

嵌入的语句不能是声明或标记语句

嵌入的语句(如 **if** 语句后面的语句)既不能包含声明, 也不能包含标记语句。

下面的示例生成 CS1023 两次:

```
// CS1023.cs
public class a
{
    public static void Main()
    {
        if (1)
            int i;      // CS1023, declaration is not valid here

        if (1)
            xx : i++; // CS1023, labeled statement is not valid here
    }
}
```

编译器错误 CS1024

错误消息

应输入预处理器指令

行以 # 符号开头, 但后面的字符串不是有效的[预处理器指令](#)。

下面的示例生成 CS1024:

```
// CS1024.cs
#import System // CS1024
```

编译器错误 CS1025

错误消息

应输入单行注释或行尾

带有[预处理器指令](#)的行不能有多行注释。

下面的示例生成 CS1025:

```
#if true /* hello
*/ // CS1025
#endif // this is a good comment
```

如果尝试使用某些无效的预处理器指令，则也可能发生 CS1025，如下所示：

```
// CS1025.cs
#define a

class Sample
{
    static void Main()
    {
        #if a 1 // CS1025, invalid syntax
            System.Console.WriteLine("Hello, World!");
        #endif
    }
}
```

编译器错误 CS1026

错误消息

应输入)

找到了不完整的语句。

发生该错误的通常原因是将语句而不是表达式放入 ASP.NET 页中的内联表达式内。例如，以下表达式是错误的：

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days;%>
```

正确的表达式应如下：

```
<%=new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days %>
```

其解释如下：

```
<% Response.Write(new TimeSpan(DateTime.Now.Ticks - new DateTime(2001, 1, 1).Ticks).Days);%>
```

下面的示例生成 CS1026：

```
// CS1026.cs
#if (a == b    // CS1026, add closing )
#endif

class x
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1027

错误消息

应输入 #endif 指令

没有为指定的 **#if** 指令找到匹配的 **#endif** 预处理器指令。或者, 在 **#if** 块中没有匹配的 **#region** 指令, 而编译器可能找到了 **#endregion** 指令。

下面的示例生成 CS1027:

```
// CS1027.cs
#if true    // CS1027, uncomment next line to resolve
// #endif

namespace x
{
    public class clk
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS1028

错误消息

意外的预处理器指令

发现了[预处理器指令](#), 但它不应出现。

例如, 发现了**#endif**, 但前面没有**#if**。

下面的示例生成 CS1028:

```
// CS1028.cs
#endif // CS1028, no matching #if
namespace x
{
    public class clk
    {
        public static void Main()
        {
    }
}
```

编译器错误 CS1029

错误消息

```
#error:"text"
```

显示用 [#error](#) 指令定义的错误文本。

下面的示例显示如何创建用户定义的错误：

```
// CS1029.cs
class Sample
{
    static void Main()
    {
        #error Let's give an error here // CS1029
    }
}
```

编译器错误 CS1031

错误消息

应输入类型

应输入类型参数。

示例

下面的示例生成 CS1031:

```
// CS1031.cs
namespace x
{
    public class ii
    {
    }

    public class a
    {
        public static operator +(a aa) // CS1031
        // try the following line instead
        // public static ii operator +(a aa)
        {
            return new ii();
        }

        public static void Main()
        {
            e = new base; // CS1031, not a type
            e = new this; // CS1031, not a type
            e = new (); // CS1031, not a type
        }
    }
}
```

编译器错误 CS1032

错误消息

不能在文件的第一个标记之后, 定义或取消定义预处理器符号

#define 和 **#undef** 预处理器指令必须用在程序的开头, 并且在其他任何关键字之前, 例如用在命名空间声明中的那些指令之前。

下面的示例生成 CS1032:

```
// CS1032.cs
namespace x
{
    public class clk
    {
        #define a    // CS1032, put before namespace
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS1033

错误消息

超出编译器限制: 文件不能超过 "number" 行

源代码文件超过了编译器可以处理的最大允许行数。若要解决该错误, 请从原始文件创建两个或多个源代码文件。最大行数为 268,435,454 行。如果使用 /debug, 则使用超过 16,707,566 行将导致调试信息损坏。

编译器错误 CS1034

错误消息

超出编译器限制: 行不能超过"number"个字符

一行中所允许的字符数限制为 16,777,214 个。

编译器错误 CS1035

错误消息

发现文件尾, 应输入“*/”

没有与注释的开始分隔符相匹配的结束分隔符。

下面的示例生成 CS1035:

```
// CS1035.cs
public class a
{
    public static void Main()
    {
    }
}
/* // CS1035, needs closing comment
```

编译器错误 CS1036

错误消息

应输入 (或 .

[doc](#) 注释中的 XML 格式错误。

编译器错误 CS1037

错误消息

应输入可重载运算符

当用 [/doc](#) 指定注释时，编译器遇到了无效链接。

编译器错误 CS1038

错误消息

应输入 #endregion 指令

#region 指令没有相匹配的 #endregion 指令。

下面的示例生成 CS1038:

```
// CS1038.cs
#region testing

public class clx
{
    public static void Main()
    {
    }
}
// CS1038
// uncomment the next line to resolve
// #endregion
```

编译器错误 CS1039

错误消息

字符串未终止

编译器检测到格式错误的 [string](#)。

示例

下面的示例生成 CS1039。若要纠正该错误, 请添加终止引号。

```
// CS1039.cs
public class MyClass
{
    public static void Main()
    {
        string b = @"hello, world; // CS1039
    }
}
```

编译器错误 CS1040

错误消息

预处理器指令必须作为一行的第一个非空白字符出现

在行上发现了[预处理器指令](#), 并且它不是该行上的第一个标记。指令必须是该行上的第一个标记。

下面的示例生成 CS1040:

```
// CS1040.cs
/* Define a symbol, X */ #define X // CS1040

// try the following two lines instead
// /* Define a symbol, X */
// #define X

public class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1041

错误消息

应输入标识符, “keyword”是关键字

在应为标识符的地方发现了 C# 语言的保留字。用用户指定的标识符替换[关键字](#)。

示例

下面的示例生成 CS1041:

```
// CS1041a.cs
class MyClass
{
    public void f(int long)    // CS1041
    // Try the following instead:
    // public void f(int i)
    {
    }

    public static void Main()
    {
    }
}
```

当从保留字设置不同的其他编程语言中导入时, 可以使用 @ 前缀修改保留标识符, 如下面的示例所示。

带 @ 前缀的标识符称作逐字标识符。

```
// CS1041b.cs
class MyClass
{
    public void f(int long)    // CS1041
    // Try the following instead:
    // public void f(int @long)
    {
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1043

错误消息

应输入 { 或 ;

属性访问器未正确声明。有关更多信息, 请参见[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1043。

```
// CS1043.cs
// compile with: /target:library
public class MyClass
{
    public int DoSomething
    {
        get return 1;    // CS1043
        set {}
    }

    // OK
    public int DoSomething2
    {
        get { return 1; }
    }
}
```

编译器错误 CS1044

错误消息

在 for、using、fixed 或声明语句中不能使用多个类型

编译器发现了无效语句。

下面的示例生成 CS1044:

```
// CS1044.cs
using System;

public class MyClass : IDisposable
{
    public void Dispose()
    {
        Console.WriteLine("Res1.Dispose()");
    }

    public static void Main()
    {
        using (MyClass mc1 = new MyClass(),
              MyClass mc2 = new MyClass()) // CS1044, remove an instantiation
        {
        }
    }
}
```

编译器错误 CS1055

错误消息

应为 add 访问器或 remove 访问器

如果未将 `event` 声明为字段, 则它必须同时定义 **add** 和 **remove** 访问器函数。

下面的示例生成 CS1055:

```
// CS1055.cs
delegate void del();
class Test
{
    public event del MyEvent
    {
        int i; // CS1055
        // uncomment accessors and delete previous line to resolve
        // add
        // {
        //     MyEvent += value;
        // }
        // remove
        // {
        //     MyEvent -= value;
        // }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1056

错误消息

意外的字符"character"

C# 编译器遇到了意外的字符，无法标识当前处理的标记。例如，如果编译器在处理标识符的过程中遇到了一个欧元字符，它将无法对该标识符进行分类，因为欧元字符只在字符串内有效，而编译器认为它没在处理字符串。

编译器错误 CS1501

错误消息

"method"方法的重载不带有"number"个参数

对类方法进行了调用，但没有一种形式的方法带有所需数目的参数。

如果正在调用所引用程序集中类上的方法，并且该方法在它的一个或多个参数上有默认值，则会发生 CS1501。C# 不能使您创建参数上带有默认值的方法，但另一种面向运行库的语言可以。如果参数（在所引用程序集的方法中）有默认值，您仍然必须调用方法并显式传递所有参数。

分配用于委托的内存时可能会发生 CS1501；您还必须指定委托所表示的方法的名称，如下面的第二个示例所示。

如果您试图实例化一个从基类继承的派生类，而该基类没有默认构造函数，但有一个带一个或多个参数的非默认构造函数，则也会发生 CS1501。若要更正此错误，请为基类提供一个默认构造函数，或使用 [base\(C# 参考\)](#) 关键字调用适当的基类构造函数，如下面的第三个示例所示。有关更多信息，请参见[使用构造函数\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1501。

```
// CS1501.cs
public class a
{
    // declare the following constructor to resolve this CS1501
    /*
    public a(int i)
    {
    }
    */
    public static void Main()
    {
        a aa = new a(2);    // CS1501
    }
}
```

下面的示例生成 CS1501。

```
// CS1501a.cs
using System;

delegate string func(int i);    // declare the delegate

class a
{
    static func dt;    // class member-field of the declared delegate type

    public static void Main()
    {
        dt = new func();    // CS1501
        // try the following line instead
        // dt = new func(z);    // this works
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
    {
        hello(8);
    }
}
```

```
    }  
}
```

下面的示例生成 CS1501。

```
// CS1501b.cs  
class Base  
{  
    public Base(string s)  
    {  
    }  
}  
  
class Derived : Base  
{ // CS1501  
    public Derived(string s)  
    {  
    }  
  
    // Try this instead:  
    // public Derived(string s) : base(s)  
    // {  
    // }  
  
    public static void Main()  
    {  
    }  
}
```

编译器错误 CS1502

错误消息

与“declaration”最匹配的重载方法具有一些无效参数

当正传递到方法的参数类型与该方法的参数类型不匹配时发生该错误。如果重载调用的方法，则重载的任何版本都不具有与正进行传递的参数类型相匹配的签名。

若要解决该问题，请执行以下操作：

- 仔细检查正进行传递的参数的类型。确保它们与正调用的方法的参数类型相匹配。
- 如果可以，请使用 [System.Convert 类](#) 转换任何不匹配的参数。
- 如果可以，请强制转换所有不匹配的参数使它们与该方法需要的类型相匹配。
- 如果可以，请定义该方法的其他重载版本以便与正要发送的参数类型相匹配。

下面的示例生成 CS1502：

```
// CS1502.cs
namespace x
{
    public class a
    {
        public a(char i)
        // try the following constructor instead
        // public a(int i)
        {
        }

        public static void Main()
        {
            a aa = new a(2222);    // CS1502
        }
    }
}
```

编译器错误 CS1503

错误消息

参数“arg”：无法从“type1”转换为“type2”

方法中某个参数的类型与实例化该类时所传递的类型不匹配。

有关如何解决该错误的讨论，请参见 [CS1502](#)。

编译器错误 CS1504

错误消息

未能打开源文件“file”(“reason”)

源文件未能由编译器打开或读取。该文件可能由其他应用程序锁定，或可能存在其他某种操作系统问题。此消息包含有关编译器无法打开或读取该文件的操作系统方面的原因。

编译器错误 CS1507

错误消息

生成模块时，无法链接资源文件“file”

在同一编译中 [/linkresource](#) 与 [/target:module](#) 一起使用，是不允许的。例如，下面的选项将生成 CS1507：

```
csc /linkresource:rf.resource /target:module in.cs
```

不过允许嵌入资源 ([/resource](#))。

编译器错误 CS1508

错误消息

此程序集中已使用了资源标识符“*identifier*”

在编译中, 相同的标识符 (*identifier*) 被传递给了两个或多个 [/resource](#) 或 [/linkresource](#) 编译器选项。

例如, 下面的选项将生成 CS1508:

```
/resource:anyfile.bmp,DuplicatIdent /linkresource:a.bmp,DuplicatIdent
```

编译器错误 CS1509

错误消息

引用的文件“file”不是程序集;请使用“/addmodule”选项代替

在使用 [/target:module](#) 的编译中(没有程序集清单)产生的输出文件(输出文件 1)被指定为了 [/reference](#)。因此, 不是程序集追加到当前程序的程序集, 而是输出文件 1 中的元数据信息将添加到当前程序的程序集。

编译器错误 CS1510

错误消息

ref 或 out 参数必须是可以赋值的变量

在方法调用中只有一个变量可以作为 ref 参数传递。ref 值与传递指针类似。

示例

下面的示例生成 CS1510:

```
// CS1510.cs
public class C
{
    public static int j = 0;

    public static void mf(ref int j)
    {
        j++;
    }

    public static void Main ()
    {
        mf (ref 2);    // CS1510, can't pass a number as a ref parameter
        // try the following to resolve the error
        // mf (ref j);
    }
}
```

编译器错误 CS1511

错误消息

关键字“base”在静态方法中不可用

在静态方法中使用了 **base** 关键字。**base** 只能在实例构造函数、实例方法或实例访问器中调用。

示例

下面的示例生成 CS1511。

```
// CS1511.cs
// compile with: /target:library
public class A
{
    public int j = 0;
}

class C : A
{
    public void Method()
    {
        base.j = 3;    // base allowed here
    }

    public static int StaticMethod()
    {
        base.j = 3;    // CS1511
        return 1;
    }
}
```

编译器错误 CS1512

错误消息

关键字“base”在当前上下文中不可用

在方法、属性或构造函数之外使用了 `base` 关键字。

下面的示例生成 CS1512:

```
// CS1512.cs
using System;

class Base {}

class CMyClass : Base
{
    private String xx = base.ToString();    // CS1512
    // Try putting this initialization in the constructor instead:
    // public CMyClass()
    // {
    //     xx = base.ToString();
    // }

    public static void Main()
    {
        CMyClass z = new CMyClass();
    }
}
```

编译器错误 CS1513

错误消息

应输入 }

编译器要求输入缺失的右大括号 ()。

下面的示例生成 CS1513:

```
// CS1513
namespace y    // CS1513, no close curly brace
{
    class x
    {
        public static void Main()
        {
    }
```

编译器错误 CS1514

错误消息

应输入 {

编译器要求输入缺失的左大括号 ({)。

下面的示例生成 CS1514:

```
// CS1514.cs
namespace x
// CS1514, no open curly brace
```

编译器错误 CS1515

错误消息

应输入 "in"

在 [foreach, in](#) 语句中，缺少 "in" 部分。

示例

下面的示例生成 CS1515：

```
using System;

class Driver
{
    static void Main()
    {
        int[] arr = new int[] {1, 2, 3};

        // try the following line instead
        // foreach (int x in arr)
        foreach (int x arr)      // CS1515, "in" is missing
        {
            Console.WriteLine(x);
        }
    }
}
```

编译器错误 CS1517

错误消息

无效的预处理器表达式

编译器遇到了无效的预处理器表达式。

有关更多信息, 请参见[预处理器指令](#)。

下面的示例显示了一些有效和无效的预处理器表达式:

```
// CS1517.cs
#if symbol      // OK
#endif
#if !symbol     // OK
#endif
#if (symbol)    // OK
#endif
#if true        // OK
#endif
#if false       // OK
#endif
#if 1           // CS1517
#endif
#if ~symbol     // CS1517
#endif
#if *           // CS1517
#endif

class x
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1518

错误消息

应输入 class、delegate、enum、interface 或 struct

发现了在 [namespace](#) 中所不支持的声明。在命名空间中，编译器只接受类、结构、枚举、接口、命名空间和委托。

示例

下面的示例生成 CS1518：

```
// CS1518.cs
namespace x
{
    sealed class c1 {};          // OK
    namespace f2 {};            // OK
    sealed f3 {};                // CS1518
}
```

编译器错误 CS1519

错误消息

类、结构或接口成员声明中的标记“token”无效

任何在类型的前面包含无效修饰符的 `class`、结构或接口的成员声明都将生成该错误。若要修复此错误，请移除无效的修饰符。

下面的示例生成 CS1519：

```
// CS1519.cs
public class IMyInterface
{
    checked void f4(); // CS1519
    // Remove "checked" from the line above.
    lock void f5(); // CS1519
    // Remove "lock" from the line above
    namespace; // CS1519
    // The line above should be removed entirely.
    int i; // OK
}
```

编译器错误 CS1520

错误消息

类、结构或接口方法必须有返回类型

在类、结构或接口中声明的方法必须具有显式返回类型。

下面的示例生成 CS1520:

```
// CS1520a.cs
public class X
{
    f7()    // CS1520, needs return type
    // try the following definition
    // void f7()
    {

    }

    public static void Main()
    {
    }
}
```

另外, 当构造函数名称的大小写与类或结构声明的不同时, 也可能会遇到该错误, 如以下示例所示:

```
// CS1520b.cs
public class Class1
{
    public class1()    // CS1520, incorrect case
    {
    }
    static void Main()
    {
    }
}
```

编译器错误 CS1521

错误消息

无效的基类型

`base` 类规范的格式不正确。

下面的示例生成 CS1521:

```
// CS1521.cs
class CMyClass
{
    public static void Main()
    {
    }
}

class CMyClass1 : CMyClass // OK
{
}

class CMyClass2 : CMyClass[] // CS1521
{
}

class CMyClass3 : CMyClass* // CS1521
{}
```

编译器错误 CS1524

错误消息

应输入 catch 或 finally

try 块后面必须跟 **catch** 或 **finally** 块。

有关异常的更多信息, 请参见 [异常处理语句\(C# 参考\)](#)。

示例

下面的示例生成 CS1524:

```
// CS1524.cs
class x
{
    public static void Main()
    {
        try
        {
            // Code here
        }
        catch
        {
        }
        try
        {
            // Code here
        }
        finally
        {
        }
        try
        {
            // Code here
        }
    }      // CS1524, missing catch or finally
}
```

编译器错误 CS1525

错误消息

无效的表达式项“character”

编译器在表达式中检测到了无效字符。

下面的示例生成 CS1525:

```
// CS1525.cs
class x
{
    public static void Main()
    {
        int i = 0;
        i = i + // OK - identifier
        'c' + // OK - character
        (5) + // OK - parenthesis
        [ + // CS1525, operator not a valid expression element
        throw + // CS1525, keyword not allowed in expression
        void; // CS1525, void not allowed in expression
    }
}
```

空白标签也可能会生成 CS1525, 如以下示例所示:

```
// CS1525b.cs
using System;
public class MyClass
{
    public static void Main()
    {
        goto FoundIt;
        FoundIt: // CS1525
        // Uncomment the following line to resolve:
        // System.Console.WriteLine("Hello");
    }
}
```

编译器错误 CS1526

错误消息

"new"表达式要求在类型后有 () 或 []

未正确指定用于为对象动态分配内存的 `new` 运算符。

示例

下面的示例显示如何使用 `new` 为数组和对象分配空间。

```
// CS1526.cs
public class y
{
    public static int i = 0;
    public int myi = 0;
}

public class z
{
    public static void Main()
    {
        y py = new y;      // CS1526
        y[] aoys = new y[10];   // Array of Ys

        for (int i = 0; i < aoys.Length; i++)
            aoys[i] = new y();   // an object of type y
    }
}
```

编译器错误 CS1527

错误消息

命名空间元素无法显式声明为 private、protected 或 protected internal

命名空间中的类型声明可以具有 [public](#) 或 [internal](#) 访问。如果没有指定可访问性，则默认是 [internal](#)。

下面的示例生成 CS1527:

```
// CS1527.cs
namespace Sample
{
    private class C1 {} // CS1527
    protected class C2 {} // CS1527
    protected internal class C3 {} // CS1527
}
```

编译器错误 CS1528

错误消息

应输入","或"="(无法在声明中指定构造函数参数)

类引用的构造就像正在创建类的对象一样。例如，尝试将变量传递给构造函数。使用 [new](#) 运算符创建类的对象。

下面的示例生成 CS1528：

```
// CS1528.cs
using System;

public class B
{
    public B(int i)
    {
        _i = i;
    }

    public void PrintB()
    {
        Console.WriteLine(_i);
    }

    private int _i;
}

public class mine
{
    public static void Main()
    {
        B b(3);    // CS1528, reference is not an object
        // try one of the following
        // B b;
        // or
        // B bb = new B(3);
        // bb.PrintB();
    }
}
```

编译器错误 CS1529

错误消息

using 子句必须位于除外部别名声明外的所有其他命名空间元素之前

using 子句在命名空间中必须最先出现。

示例

下面的示例生成 CS1529:

```
// CS1529.cs
namespace X
{
    namespace Subspace
    {
        using Microsoft;

        class SomeClass
        {
        };

        using Microsoft;      // CS1529, place before class definition
    }

    using System.Reflection; // CS1529, place before namespace 'Subspace'
}

using System;           // CS1529, place at the beginning of the file
```

编译器错误 CS1530

错误消息

命名空间元素上不允许使用关键字 new

没有必要在 [namespace](#) 中的任何构造上均指定 new 关键字。

下面的示例生成 CS1530:

```
// CS1530.cs
namespace a
{
    new class i    // CS1530
    {
    }

    // try the following instead
    class ii
    {
        public static void Main()
        {
        }
    }
}
```

编译器错误 CS1534

错误消息

重载的二元运算符“operator”接受两个参数

二进制可重载的运算符的定义必须带两个参数。

下面的示例生成 CS1534:

```
// CS1534.cs
class MyClass
{
    public static MyClass operator - (MyClass MC1, MyClass MC2, MyClass MC3)    // CS1534
    // try the following line instead
    // public static MyClass operator - (MyClass MC1, MyClass MC2)
    {
        return new MyClass();
    }

    public static int Main()
    {
        return 1;
    }
}
```

编译器错误 CS1535

错误消息

重载的一元运算符“operator”接受一个参数

一元可重载的运算符的定义必须带一个参数。

示例

下面的示例生成 CS1535:

```
// CS1535.cs
class MyClass
{
    // uncomment the method parameter to resolve CS1535
    public static MyClass operator ++ /*MyClass MC1*/ // CS1535
    {
        return new MyClass();
    }

    public static int Main()
    {
        return 1;
    }
}
```

编译器错误 CS1536

错误消息

参数类型 void 无效

指定 void 指针以外的 void 参数没有必要或者是无效的。

下面的示例生成 CS1536:

```
// CS1536.cs
class a
{
    public static int x( void )    // CS1536
    // try the following line instead
    // public static int x()
    {
        return 0;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1537

错误消息

using 别名“alias”以前在此命名空间中出现过

您两次将某个符号定义为了命名空间的别名。一个符号只能定义一次。

下面的示例生成 CS1537:

```
// CS1537.cs
namespace x
{
    using System;
    using Object = System.Object;
    using Object = System.Object;    // CS1537, delete this line to resolve
    using System = System;
}
```

编译器错误 CS1540

错误消息

无法通过类型“type1”的限定符访问保护成员“member”；限定符必须是类型“type2”(或者从该类型派生的)

虽然派生的 `class` 可以访问其基类的受保护成员，但它无法通过基类的实例这样做。

下面的示例生成 CS1540：

```
// CS1540.cs
public class Base
{
    protected void func()
    {
    }
}

public class Derived : Base
{
    public static void test(Base anotherInstance)
        // the method declaration could be changed as follows
        // public static void test(Derived anotherInstance)
    {
        anotherInstance.func();    // CS1540
    }
}

public class Tester : Derived
{
    public static void Main()
    {
        Base pBase = new Base();
        // the allocation could be changed as follows
        // Derived pBase = new Derived();
        test(pBase);
    }
}
```

编译器错误 CS1541

错误消息

无效的引用选项: "symbol"—无法引用目录

编译器检测到指定目录而不是特定文件的尝试。例如，当使用 [/reference](#) 编译器选项时，必须指定文件；指定目录是不可能的。

例如，向编译器传递 `/reference:c:\` 将生成 CS1541。

编译器错误 CS1542

错误消息

"`dll`"无法添加到此程序集中, 因为它已经是程序集; 请使用"/R"选项

用 `/addmodule` 编译器选项引用的文件不是用 `/target:module` 生成的; 请在该编译中使用 `/reference` 引用文件。

编译器错误 CS1545

错误消息

属性、索引器或事件“property”不受该语言支持;请尝试直接调用访问器方法“set accessor”或“get accessor”

您的代码正在使用具有默认[索引器](#)的对象并试图使用索引语法。若要解决此错误，请调用属性的 **get** 或 **set** 访问器方法。

示例

```
// CPP1545.cpp
// compile with: /clr /LD
// a Visual C++ program
using namespace System;
public ref struct Employee {
    Employee( String^ s, int d ) {}

    property String^ name {
        String^ get() {
            return nullptr;
        }
    }
};

public ref struct Manager {
    property Employee^ Report [String^] {
        Employee^ get(String^ s) {
            return nullptr;
        }

        void set(String^ s, Employee^ e) {}
    }
};
```

下面的示例生成 CS1545。

```
// CS1545.cs
// compile with: /r:CPP1545.dll

class x {
    public static void Main() {
        Manager Ed = new Manager();
        Employee Bob = new Employee("Bob Smith", 12);
        Ed.Report( Bob.name ) = Bob;    // CS1545
        Ed.set_Report( Bob.name, Bob);   // OK
    }
}
```

编译器错误 CS1546

错误消息

属性、索引器或事件“property”不受该语言支持;请尝试直接调用访问器方法“accessor”

代码正在使用具有默认索引属性的对象，并试图使用索引语法。若要解决该错误，请调用属性的访问器方法。有关索引器和属性的更多信息，请参见[索引器\(C# 编程指南\)](#)和[定义和使用属性](#)。

下面的示例生成 CS1546。

示例

此代码示例由一个编译为 .dll 的 .cpp 文件和一个使用该 .dll 的 .cs 文件组成。下面是 .dll 文件的代码，它定义了一个将由 .cs 文件中的代码访问的属性。

```
// CPP1546.cpp
// compile with: /clr /LD
using namespace System;
public ref class MCPP
{
public:
    property int Prop [int,int]
    {
        int get( int i, int b )
        {
            return i;
        }
    }
};
```

这是 C# 文件。

```
// CS1546.cs
// compile with: /r:CPP1546.dll
using System;
public class Test
{
    public static void Main()
    {
        int i = 0;
        MCPP mcpp = new MCPP();
        i = mcpp.Prop(1,1); // CS1546
        // Try the following line instead:
        // i = mcpp.get_Prop(1,1);
    }
}
```

编译器错误 CS1547

错误消息

关键字“void”不能用于此上下文中

编译器检测到无效的 `void` 关键字使用。

下面的示例生成 CS1547:

```
// CS1547.cs
public class MyClass
{
    void BadMethod()
    {
        void i;    // CS1547, cannot have variables of type void
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1548

错误消息

对程序集“assembly”签名时加密失败 —“reason”

程序集签名失败时将发生 CS1548。这通常是由于无效的密钥文件名称、无效的密钥文件路径或已损坏的密钥文件而造成的。

若要对程序集进行完全签名，必须提供一个包含有关公钥和私钥信息的有效密钥文件。若要延迟为程序集签名，必须选中“仅延迟签名”复选框并提供一个包含有关公钥信息的有效密钥文件。为程序集延迟签名时不需要提供私钥。

有关更多信息，请参见[如何:对程序集进行签名 \(Visual Studio\)](#)、[/keyfile \(指定强名称密钥文件\) \(C# 编译器选项\)](#)和[/delaysign \(延迟为程序集签名\) \(C# 编译器选项\)](#)。

创建程序集时，C# 编译器会调入一个称为“al.exe”的实用工具。如果在创建程序集时出现失败，al.exe 会报告失败的原因。请参见[Al.exe 工具错误和警告](#)并在该主题中搜索编译器在“原因”中报告的文本。

请参见

任务

[如何:对程序集进行签名 \(Visual Studio\)](#)

编译器错误 CS1549

错误消息

未找到合适的加密服务

未安装 [RSA](#) 兼容的加密服务提供程序。

编译器错误 CS1551

错误消息

索引器必须至少有一个参数

声明的索引器不带参数。

下面的示例生成 CS1551:

```
// CS1551.cs
public class MyClass
{
    int intI;

    int this[] // CS1551
    // try the following line instead
    // int this[int i]
    {
        get
        {
            return intI;
        }
        set
        {
            intI = value;
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1552

错误消息

数组类型说明符 [] 必须出现在参数名之前

数组类型说明符在数组声明中的位置是在变量名之后。

示例

下面的示例生成 CS1552:

```
// CS1552.cs
public class C
{
    public static void Main(string args[])    // CS1552
    // try the following line instead
    // public static void Main(string [] args)
    {
    }
}
```

编译器错误 CS1553

错误消息

声明无效;请改用“modifier operator <dest-type> (...”

[operator](#) 的返回类型必须紧挨在参数列表之前, 并且 *modifier* 为 **implicit** 或 **explicit**。

下面的示例生成 CS1553:

```
// CS1553.cs
class MyClass
{
    public static int implicit operator (MyClass f) // CS1553
    // try the following line instead
    // public static implicit operator int (MyClass f)
    {
        return 6;
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1554

错误消息

声明无效;请改用“<type> operator op (...”

用户定义的 [operator](#) 的返回类型必须出现在关键字运算符之前。

下面的示例生成 CS1554:

```
// CS1554.cs
class MyClass
{
    public static operator ++ MyClass (MyClass f)      // CS1554
    // try the following line instead
    // public static MyClass operator ++ (MyClass f)
    {
        return new MyClass ();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1555

错误消息

未能找到为 Main 方法指定的“class”

给 [/main](#) 编译器选项指定了类，但在源代码中没有找到该类的名称。

编译器错误 CS1556

错误消息

为 Main 方法指定的“construct”必须是有效的类或结构

给 [/main](#) 编译器选项传递的标识符不是类名。

编译器错误 CS1557

错误消息

无法对 Main 方法使用“class”，因为它在不同的输出文件中

为多输出文件编译中的一个输出文件指定了 [/main](#) 编译器选项。但是，在 /main 编译的源代码中没有找到该类，却在编译中的其他某个输出文件的源代码文件中找到了它。

编译器错误 CS1558

错误消息

"class"没有合适的静态 Main 方法

/main 编译器选项指定了一个在其中查找 **Main** 方法的类。但是, **Main** 方法未正确定义。

由于返回类型无效, 下面的示例生成 CS1558。

```
// CS1558.cs
// compile with: /main:MyNamespace.MyClass

namespace MyNamespace
{
    public class MyClass
    {
        public static float Main()
        {
            return 0.0; // CS1558 because the return type is a float.
        }
    }
}
```

编译器错误 CS1559

错误消息

"object"被导入, 不能用作程序"program"的入口点

给 [/main](#) 编译器选项指定的类无效, 该类不能用作 [Main](#) 方法的位置。

编译器错误 CS1560

错误消息

为预处理器指令指定的文件名无效。文件名太长，或者不是有效的文件名

用 #line(C# 参考) 指定的文件名超过了 _MAX_PATH(256 个字符), 或者 #line 所在的行超过了 2000 个字符。

示例

下面的示例生成 CS1560。

编译器错误 CS1561

错误消息

输出的文件名太长或无效

输出文件名的长度不能超过 256 个字符，并且不得包含无效的字符，例如欧元符号、问号，或反斜杠及其他。

编译器错误 CS1562

错误消息

必须为没有源的输出指定 /out 选项

编译可以创建输出文件，但没有作为输入的源代码文件(输出文件的名称可以暗含在其中)。例如，您可能正在试图编译纯元数据文件或纯资源文件。

使用 [/out](#) 编译器选项指定输出文件的名称。

编译器错误 CS1563

错误消息

输出“output file”没有任何源文件

指定了 [/out](#) 编译器选项，但其后没有源代码文件。您既不应使用 **/out**，也不应在 **/out** 的后面指定源代码文件。

编译器错误 CS1565

错误消息

指定的选项冲突: Win32 资源文件; Win32 图标

在同一编译中同时指定 [/win32res](#) 编译器选项和 [/win32ico](#) 编译器选项无效。

编译器错误 CS1566

错误消息

读取资源文件“file”时出错 —“reason”

编译器在将文件名传递给 [/resource](#) 编译器选项时遇到麻烦。

编译器错误 CS1567

错误消息

生成 Win32 资源时出错：“file”

您的编译或者使用了 [/win32icon](#) 编译器选项，或者未使用 [/win32res](#) 编译器选项，这导致编译器生成包含资源信息的文件，但编译器因磁盘空间不足或其他某些错误而无法创建文件。

如果您无法解决此文件生成问题，可以使用 [/win32res](#)，该编译器选项不生成包含资源信息的文件。

编译器错误 CS1569

错误消息

生成 XML 文档文件“Filename”时出错(“reason”)

当试图将 XML 文档写入消息中提到的文件时，由于指定的原因，会出现错误。这可能是由于类似“未找到网络驱动器”或“访问被拒绝”等原因导致的。通常情况下，错误原因中会给出为更正此错误需采取哪些操作的建议。例如，如果错误中指出“访问被拒绝”，您应当验证对文件是否有写权限。

示例

```
// CS1569.cs
// compile with: /doc:CS1569.xml
// post-build command: attrib +r CS1569.xml
class Test
{
    /// <summary>Test.</summary>
    public static void Main() {}
}
```

上面的示例生成一个当时设置为只读的 .xml 文件。此示例试图写入同一文件。下面的示例生成 CS1569。

```
// CS1569_a.cs
// compile with: /doc:CS1569.xml
// CS1569 expected
class Test
{
    /// <summary>Test.</summary>
    public static void Main() {}
}
```

编译器错误 CS1575

错误消息

stackalloc 表达式在类型后要求有 []

使用 [stackalloc](#) 请求的分配大小必须在方括号中指定。

下面的示例生成 CS1575:

```
// CS1575.cs
// compile with: /unsafe
public class MyClass
{
    unsafe public static void Main()
    {
        int *p = stackalloc int (30);    // CS1575
        // try the following line instead
        // int *p = stackalloc int [30];
    }
}
```

编译器错误 CS1576

错误消息

为 #line 指令指定的行号缺少或无效

编译器检测到传递给 `#line` 指令的值有错误。

下面的示例生成 CS1576:

```
// CS1576.cs
public class MyClass
{
    static void Main()
    {
        #line "abc.sc"          // CS1576
        // try the following line instead
        //#line 101 "abc.sc"
        intt i; // error will be reported on line 101
    }
}
```

编译器错误 CS1577

错误消息

程序集生成失败 —

编译的程序集生成部分失败。有关更多信息, 请参见 alink 实用工具的错误文档([Al.exe 工具错误和警告](#))。

编译器错误 CS1578

错误消息

应输入文件名、单行注释或行尾

在 `#line` 指令后只能有一个文件名(在双引号中)或单行注释。

下面的示例生成 CS1578:

```
// CS1578.cs
class MyClass
{
    static void Main()
    {
        #line 101 abc.cs    // CS1578
        // try the following line instead
        //#line 101 "abc.cs"
        intt i;           // error will be reported on line 101
    }
}
```

编译器错误 CS1579

错误消息

`foreach` 语句不能对“`type1`”类型的变量进行操作，因为“`type2`”不包含“`identifier`”的公共定义

若要使用 `foreach` 语句循环访问某个集合，该集合必须满足以下要求：

- 它必须是一个接口、类或结构。
- 它必须包含一个返回类型的公共 `GetEnumerator` 方法。
- 返回类型必须包含一个名为 `Current` 的公共属性和一个名为 `MoveNext` 的公共方法。
- 有关更多信息，请参见[如何：使用 foreach 访问集合类\(C# 编程指南\)](#)。

示例

在本示例中，`foreach` 无法循环访问集合，因为 `MyCollection` 集合中没有 `public GetEnumerator` 方法。

下面的示例生成 CS1579。

```
// CS1579.cs
using System;
public class MyCollection
{
    int[] items;
    public MyCollection()
    {
        items = new int[5] {12, 44, 33, 2, 50};
    }

    // Delete the following line to resolve.
    MyEnumerator GetEnumerator()

    // Uncomment the following line to resolve:
    // public MyEnumerator GetEnumerator()
    {
        return new MyEnumerator(this);
    }

    // Declare the enumerator class:
    public class MyEnumerator
    {
        int nIndex;
        MyCollection collection;
        public MyEnumerator(MyCollection coll)
        {
            collection = coll;
            nIndex = -1;
        }

        public bool MoveNext()
        {
            nIndex++;
            return(nIndex < collection.items.GetLength(0));
        }

        public int Current
        {
            get
            {
                return(collection.items[nIndex]);
            }
        }
    }

    public static void Main()
```

```
{  
    MyCollection col = new MyCollection();  
    Console.WriteLine("Values in the collection are:");  
    foreach (int i in col) // CS1579  
    {  
        Console.WriteLine(i);  
    }  
}
```

编译器错误 CS1583

错误消息

"file"不是有效的 Win32 资源文件

传递给 [/win32res](#) 编译器选项的文件不是有效的资源文件。

编译器错误 CS1585

错误消息

成员修饰符“keyword”必须位于成员类型和名称之前

方法签名中的访问说明符没有出现在正确的位置。

下面的示例生成 CS1585:

```
// CS1585.cs
public class Class1
{
    public void static Main(string[] args)    // CS1585
    // try the following line instead
    // public static void Main(string[] args)
    {
    }
}
```

编译器错误 CS1586

错误消息

数组创建必须有数组大小或数组初始值设定项

数组未正确声明。

下面的示例生成 CS1586:

```
// CS1586.cs
using System;
class MyClass
{
    public static void Main()
    {
        int[] a = new int[];    // CS1586
        // try the following line instead
        int[] b = new int[5];
    }
}
```

编译器错误 CS1588

错误消息

无法确定公共语言运行库目录 --"reason"

编译器无法检索到 .NET Framework 公共语言运行库的安装目录。该错误指示公共语言运行库未正确安装。

编译器错误 CS1593

错误消息

委托“del”不接受“number”参数

传递给 [delegate](#) 调用的参数数目与委托声明中的参数数目不一致。

下面的示例生成 CS1593:

```
// CS1593.cs
using System;
delegate string func(int i); // declare delegate

class a
{
    public static void Main()
    {
        func dt = new func(z);
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
    {
        hello(8, 9); // CS1593
        // try the following line instead
        // hello(8);
    }
}
```

编译器错误 CS1594

错误消息

委托“delegate”有一些无效参数

传递给 [delegate](#) 调用的参数类型与委托声明中的参数类型不一致。

下面的示例生成 CS1594:

```
// CS1594.cs
using System;
delegate string func(int i);    // declare delegate

class a
{
    public static void Main()
    {
        func dt = new func(z);
        x(dt);
    }

    public static string z(int j)
    {
        Console.WriteLine(j);
        return j.ToString();
    }

    public static void x(func hello)
    {
        hello("8");    // CS1594
        // try the following line instead
        // hello(8);
    }
}
```

编译器错误 CS1597

错误消息

方法或访问器块后面的分号无效

不需要(或不允许)使用分号结束方法或访问器块。

下面的示例生成 CS1597:

```
// CS1597.cs
class TestClass
{
    public static void Main()
    {
        }; // CS1597, remove semicolon
}
```

编译器错误 CS1599

错误消息

方法或委托不能返回“type”类型

基类库中的某些类型，例如 [TypedReference](#) 和 [ArgIterator](#) 不能用作返回类型。

下面的示例生成 CS1599：

```
// CS1599.cs
using System;

class MyClass
{
    public static void Main()
    {

        public TypedReference Test1() // CS1599
        {
            return null;
        }

        public ArgIterator Test2() // CS1599
        {
            return null;
        }
}
```

编译器错误 CS1600

错误消息

编译被用户取消

当使用 Visual Studio IDE 时, 取消了使用 C# 编译器的编译。

编译器错误 CS1601

错误消息

方法或委托参数不能是“type”类型

基类库中的某些类型，例如 [TypedReference](#) 和 [ArgIterator](#) 不能与 `ref` 或 `out` 关键字一起用作的参数类型。

下面的示例生成 CS1601：

```
// CS1601.cs
using System;

class MyClass
{
    public void Test1 (ref TypedReference t)    // CS1601
    {
    }

    public void Test2 (out ArgIterator t)    // CS1601
    {
    }
}
```

编译器错误 CS1604

错误消息

无法给“variable”赋值，因为它是只读的

对只读变量进行了赋值。若要避免此错误，不要在此上下文中给此变量赋值或试图修改此变量。

编译器错误 CS1605

错误消息

无法将“var”作为 ref 或 out 参数传递，因为它是只读的

作为 **ref** 或 **out** 参数传递的变量应在调用的方法中修改。因此，不可能将只读参数作为 **ref** 或 **out** 传递。

编译器错误 CS1606

错误消息

程序集签名失败; 不能对输出签名 — reason

产生了程序集, 但当编译器试图完成对它的签名时出现失败。

编译器错误 CS1608

错误消息

C# 类型上不允许有 Required 属性

在 C# 代码中定义的类型上不允许 [RequiredAttributeAttribute](#)。

编译器错误 CS1609

错误消息

修饰符不能放置在事件访问器声明中

修饰符只能放置在事件声明中，不能放置在事件访问器声明中。有关更多信息，请参见[使用属性\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1609。

```
// CS1609.cs
// compile with: /target:library
delegate int Del();
class A
{
    public event Del MyEvent
    {
        private add {}    // CS1609
        // try the following line instead
        // add {}
        remove {}
    }
}
```

编译器错误 CS1611

错误消息

params 参数不能声明为 ref 或 out

关键字 ref 或 out 不能与 params 关键字一起使用。

下面的示例生成 CS1611:

```
// CS1611.cs
public class MyClass
{
    public static void Test(params ref int[] a)    // CS1611, remove ref
    {
    }

    public static void Main()
    {
        Test(1);
    }
}
```

编译器错误 CS1612

错误消息

无法修改“expression”的返回值，因为它不是变量

试图修改的值类型是中间表达式的结果。该值是不持久的，因此它将保持不变。

若要解决该错误，请将表达式的结果存储在中间值中，或对中间表达式使用引用类型。

示例

下面的代码生成错误 CS1612。

```
// CS1612.cs
public struct MyStruct
{
    public int Width;
}

public class ListView
{
    public MyStruct Size
    {
        get { return new MyStruct(); }
    }
}

public class MyClass
{
    public MyClass()
    {
        ListView lvi;
        lvi = new ListView();
        lvi.Size.Width = 33; // CS1612
        // Use this instead:
        // MyStruct temp = lvi.Size;
        // temp.Width = 33;
    }

    public static void Main() {}
}
```

编译器错误 CS1613

错误消息

无法找到接口“interface”的托管 coclass 包装类“class”(是否缺少程序集引用?)

试图从接口实例化 COM 对象。该接口具有 **ComImport** 和 **CoClass** 属性，但编译器无法找到为 **CoClass** 属性提供的类型。

若要解决此错误，可以尝试使用下列方法之一：

- 向具有该 coclass 的程序集添加引用(多数情况下，该接口和该 coclass 应该在同一程序集中)。有关信息，请参见 [/reference](#) 或[“添加引用”对话框](#)。
- 修复该接口上的 **CoClass** 属性。

下面的示例演示了 **CoClassAttribute** 的正确用法：

```
// CS1613.cs
using System;
using System.Runtime.InteropServices;

[Guid("1FFD7840-E82D-4268-875C-80A160C23296")]
[ComImport()]
[CoClass(typeof(A))]
public interface IA{}
public class A : IA {}

public class AA
{
    public static void Main()
    {
        IA i;
        i = new IA(); // This is equivalent to new A().
                      // because of the CoClass attribute on IA
    }
}
```

编译器错误 CS1614

错误消息

"attribute"不明确, 请使用"@attribute"或"attributeAttribute"

编译器遇到了意义不明确的属性指定。

为简便起见, C# 编译器允许将 **ExampleAttribute** 指定为 [Example]。不过, 如果名为 **Example** 的属性类与 **ExampleAttribute** 同时存在, 则会引起多义性, 因为编译器无法判断 [Example] 是指 **Example** 属性还是指 **ExampleAttribute** 属性。要解决此问题, 请对 **Example** 属性使用 [@Example], 而对 **ExampleAttribute** 使用 [ExampleAttribute]。

下面的示例生成 CS1614:

```
// CS1614.cs
using System;

// Both of the following classes are valid attributes with valid
// names (MySpecial and MySpecialAttribute). However, because the lookup
// rules for attributes involves auto-appending the 'Attribute' suffix
// to the identifier, these two attributes become ambiguous; that is,
// if you specify MySpecial, the compiler can't tell if you want
// MySpecial or MySpecialAttribute.

public class MySpecial : Attribute {
    public MySpecial() {}
}

public class MySpecialAttribute : Attribute {
    public MySpecialAttribute() {}
}

class MakeAWarning {
    [MySpecial()] // CS1614
                  // Ambiguous: MySpecial or MySpecialAttribute?
    public static void Main() {
    }

    [@MySpecial()] // This isn't ambiguous, it binds to the first attribute above.
    public static void NoWarning() {
    }

    [MySpecialAttribute()] // This isn't ambiguous, it binds to the second attribute above.
    public static void NoWarning2() {
    }

    [@MySpecialAttribute()] // This is also legal.
    public static void NoWarning3() {
    }
}
```

编译器错误 CS1615

错误消息

参数“number”不应与关键字“keyword”一起传递

当函数没有接受 **ref** 或 **out** 参数作为该参数时，使用了关键字 **ref** 或 **out** 中的一个。若要解决此错误，请移除不正确的关键字，并使用与函数声明匹配的正确的关键字(如果有的话)。

下面的示例生成 CS1615：

```
// CS1615.cs
class C
{
    public void f(int i) {}
    public static void Main()
    {
        int i = 1;
        f(ref i); // CS1615
    }
}
```

编译器错误 CS1617

错误消息

/langversion 的选项“option”无效;必须是 ISO-1 或 Default

如果您使用了 [/langversion](#) 命令行开关或项目设置, 但没有指定一个有效的语言选项, 则会出现此错误。要解决此错误, 请检查命令行语法或项目设置, 将其更改为其中一个列出的选项。

例如, 使用 csc /langversion:ISO 编译将生成错误 CS1617。

编译器错误 CS1618

错误消息

无法用“method”创建委托，因为它具有 Conditional 属性

无法用条件方法创建委托，因为在某些版本中可能不存在该方法。

下面的示例生成 CS1618：

```
// CS1618.cs
using System;
using System.Diagnostics;

delegate void del();

class MakeAnError {
    public static void Main() {
        del d = new del(ConditionalMethod); // CS1618
        // Invalid because on builds where DEBUG is not set,
        // there will be no "ConditionalMethod".
    }
    // To fix the error, remove the next line:
    [Conditional("DEBUG")]
    public static void ConditionalMethod()
    {
        Console.WriteLine("Do something only in debug");
    }
}
```

编译器错误 CS1619

错误消息

无法创建临时文件“filename”— reason

由于特定的原因(例如, 磁盘已满), 编译器无法创建临时文件。

编译器错误 CS1620

错误消息

参数“number”必须与关键字“keyword”一起传递

如果您要将参数传递给带有 **ref** 或 **out** 参数的函数，但您在调用时没有包括 **ref** 或 **out** 关键字，或者包括了错误的关键字，就会发生此错误。错误文本中指出了应使用的正确关键字以及导致失败的参数。

下面的示例生成 CS1620：

```
// CS1620.cs
class C
{
    void f(ref int i) {}
    public static void Main()
    {
        int x = 1;
        f(out x); // CS1620 - f takes a ref parameter, not an out parameter
        // Try this line instead:
        // f(ref x);
    }
}
```

编译器错误 CS1621

错误消息

yield 语句不能在匿名方法块内使用

yield 语句不能存在于迭代器的匿名方法块中。

示例

下面的示例生成 CS1621:

```
// CS1621.cs

using System.Collections;

delegate object MyDelegate();

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        MyDelegate d = delegate
        {
            yield return this; // CS1621
            return this;
        };
        d();
        // Try this instead:
        // MyDelegate d = delegate { return this; };
        // yield return d();
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1622

错误消息

无法从迭代器返回值。请使用 yield return 语句返回值，或使用 yield break 语句结束迭代。

迭代器是一个特殊的函数，它通过 yield 语句而不是 return 语句返回值。有关更多信息，请参见[迭代器](#)。

下面的示例生成 CS1622：

```
// CS1622.cs
// compile with: /target:library
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        return (IEnumerator) this; // CS1622
        yield return this; // OK
    }
}
```

编译器错误 CS1623

错误消息

迭代器不能有 **ref** 或 **out** 参数

如果迭代器方法带有 **ref** 或 **out** 参数，则会发生此错误。若要避免此错误，请从方法签名中移除 **ref** 或 **out** 关键字。

示例

下面的示例生成 CS1623：

```
// CS1623.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        yield return 0;
    }

    // To resolve the error, remove ref
    public IEnumerator GetEnumerator(ref int i) // CS1623
    {
        yield return i;
    }

    // To resolve the error, remove out
    public IEnumerator GetEnumerator(out float f) // CS1623
    {
        f = 0.0F;
        yield return f;
    }
}
```

编译器错误 CS1624

错误消息

"accessor"体不能是迭代器块, 因为"type"不是迭代器接口类型

当使用了迭代器访问器, 但返回类型不是以下迭代器接口类型之一时会发生此错误:[IEnumerable](#)、[IEnumearable](#)、[IEnumerator](#)和[IEnumerator](#)。若要避免此错误, 请将其中一个迭代器接口类型用作返回类型。

示例

下面的示例生成 CS1624:

```
// CS1624.cs
using System;
using System.Collections;

class C
{
    public int Iterator
    // Try this instead:
    // public IEnumerable Iterator
    {
        get // CS1624
        {
            yield return 1;
        }
    }
}
```

编译器错误 CS1625

错误消息

无法在 finally 子句体内生成

在 finally 子句体内不允许出现 yield 语句。若要避免此错误，请将 yield 语句移到 finally 子句的外部。

下面的示例生成 CS1625:

```
// CS1625.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        try
        {
        }
        finally
        {
            yield return this; // CS1625
        }
    }

    public class CMain
    {
        public static void Main() { }
    }
}
```

编译器错误 CS1626

错误消息

无法在包含 catch 子句的 try 块体内生成值

如果有一个 catch 子句与 try 块关联，该 try 块内不允许有 yield 语句。若要避免此错误，请将 yield 语句移到 catch 子句的外部。

下面的示例生成 CS1626：

```
// CS1626.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        try
        {
            yield return this; // CS1626
        }
        catch
        {

        }
    }

    public class CMain
    {
        public static void Main() { }
    }
}
```

编译器错误 CS1627

错误消息

yield return 后应为表达式

如果使用 **yield** 时未包含表达式，则会发生此错误。若要避免此错误，请在语句中插入适当的表达式。

下面的示例生成 CS1627:

```
// CS1627.cs
using System.Collections;

class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        yield return;    // CS1627
        // To resolve, add the following line:
        // yield return 0;
    }
}

public class CMain
{
    public static void Main() { }
```

编译器错误 CS1628

错误消息

不能在匿名方法块内使用 ref 或 out 参数“parameter”

如果您在匿名方法块内使用 ref 或 out 参数，则会出现此错误。若要避免此错误，请使用局部变量或某个其他的构造。

下面的示例生成 CS1628：

```
// CS1628.cs

delegate int MyDelegate();

class C
{
    public static void F(ref int i)
    {
        MyDelegate d = delegate { return i; }; // CS1628
        // Try this instead:
        // int tmp = i;
        // MyDelegate d = delegate { return tmp; };
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1629

错误消息

迭代器中不能出现不安全的代码。

C# 语言规范不允许迭代器中出现不安全的代码。

下面的示例生成 CS1629:

```
// CS1629.cs
// compile with: /unsafe
using System.Collections.Generic;
class C
{
    IEnumerator<int> IteratorMeth() {
        int i;
        unsafe // CS1629
        {
            int *p = &i;
            yield return *p;
        }
    }
}
```

编译器错误 CS1630

错误消息

/errorreport 的选项“option”无效; 必须是 prompt、send、queue 或 none

命令行选项 [/errorreport](#) 后面必须跟随 **prompt**、**send**、**queue** 或 **none**, 用于指定当发生内部编译器错误时要采取的操作。

编译器错误 CS1631

错误消息

无法在 catch 子句体内生成值

不允许从 catch 子句体内使用 yield 语句。若要避免此错误, 请将 yield 语句移到 catch 子句体的外面。

下面的示例生成 CS1631:

```
// CS1631.cs
using System;
using System.Collections;

public class C : IEnumerable
{
    public IEnumerator GetEnumerator()
    {
        try
        {
        }
        catch(Exception e)
        {
            yield return this; // CS1631
        }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1632

错误消息

控制不能离开匿名方法体

如果跳转语句(**break**、**goto**、**continue** 等)试图将控制移到匿名方法块以外，则会发生此错误。匿名方法块是一个函数体，只能通过返回语句或到达块的末尾退出。

下面的示例生成 CS1632:

```
// CS1632.cs
// compile with: /target:library
delegate void MyDelegate();
class MyClass
{
    public void Test()
    {
        for (int i = 0 ; i < 5 ; i++)
        {
            MyDelegate d = delegate {
                break;    // CS1632
            };
        }
    }
}
```

编译器错误 CS1637

错误消息

迭代器不能有不安全的参数或 yield 类型

检查迭代器的参数列表和任何 yield 语句的类型以验证您没有使用任何不安全的类型。

示例

下面的示例生成 CS1637:

```
// CS1637.cs
// compile with: /unsafe
using System.Collections;

public unsafe class C
{
    public IEnumrator Iterator1(int* p) // CS1637
    {
        yield return null;
    }
}
```

编译器错误 CS1638

错误消息

"identifier"是一个保留标识符，不能在使用 ISO 语言版本模式时使用

当 ISO 语言兼容性选项由 **/langversion** 编译器开关指定时，任何含有双下划线的标识符都会产生此错误。若要避免此错误，请消除任何含有双下划线的标识符，或者不使用 ISO-1 语言版本选项。

示例

下面的示例生成 CS1638：

```
// CS1638.cs
// compile with: /langversion:ISO-1
class bad__identifier // CS1638 (double underscores are not ISO compliant)
{
}

// Try this instead:
//class GoodIdentifier
//{
//}

class CMain
{
    public static void Main() { }
```

请参见

参考

[/langversion\(符合规范的语法\)\(C# 编译器选项\)](#)

编译器错误 CS1639

错误消息

接口“interface”的托管 coclass 包装类签名“signature”不是有效的类名签名。

当接口的元数据无效时会发生此错误。如果此接口是使用 C# 或受支持的 .NET 语言生成的，则永远不会发生此错误。请与接口的供应商核实或验证接口程序集是否已成功生成。

编译器错误 CS1640

错误消息

foreach 语句不能对“type”类型的变量进行操作，因为它实现“interface”的多个实例化，请尝试强制转换为特定的接口实例化

此类型从 IEnumarator<T> 的两个或更多的实例继承，这意味着此类型不存在 **foreach** 可以使用的唯一枚举。指定 IEnumarator<T> 的类型，或使用其他循环构造。

示例

下面的示例生成 CS1640：

```
// CS1640.cs

using System;
using System.Collections;
using System.Collections.Generic;

public class C : IEnumerable, IEnumerable<int>, IEnumerable<string>
{
    IEnumarator<int> IEnumerable<int>.GetEnumarator()
    {
        yield break;
    }

    IEnumarator<string> IEnumerable<string>.GetEnumarator()
    {
        yield break;
    }

    IEnumarator IEnumerable.GetEnumarator()
    {
        return (IEnumarator)((IEnumerable<string>)this).GetEnumarator();
    }
}

public class Test
{
    public static int Main()
    {
        foreach (int i in new C()){} // CS1640

        // Try specifying the type of IEnumarator<T>
        // foreach (int i in (IEnumarator<int>)new C()){}
        return 1;
    }
}
```

编译器错误 CS1641

错误消息

固定大小缓冲区字段的字段名称后面必须有数组大小说明符

与规则数组不同，固定大小缓冲区要求在声明点指定固定大小。若要解决此错误，请添加一个正整数字符或正整数常数，并在标识符后面加上方括号。

下面的示例生成 CS1641：

```
// CS1641.cs
// compile with: /unsafe /target:library
unsafe struct S {
    fixed int [] a; // CS1641

    // OK
    fixed int b [10];
    const int c = 10;
    fixed int d [c];
}
```

编译器错误 CS1642

错误消息

固定大小的缓冲区字段只能是结构的成员。

如果您在 **class** 而不是 **struct** 中使用固定大小的缓冲区，将会出现此错误。若要解决此错误，请将 **class** 更改为 **struct** 或者将字段声明为普通数组。

示例

下面的示例生成 CS1642。

```
// CS1642.cs
// compile with: /unsafe /target:library
unsafe class C
{
    fixed int a[10];    // CS1642
}

unsafe struct D
{
    fixed int a[10];
}

unsafe class E
{
    public int[] a = null;
}
```

编译器错误 CS1643

错误消息

"type"类型的匿名方法:并非所有的代码路径都返回值

如果一个委托体没有返回语句, 或者它的返回语句无法由编译器访问, 则会出现此错误。在下面的示例中, 编译器不尝试预测分支条件的结果以验证匿名方法块是否始终返回一个值。

示例

下面的示例生成 CS1643:

```
// CS1643.cs
delegate int MyDelegate();

class C
{
    static void Main()
    {
        MyDelegate d = delegate
        {
            int i = 0;
            if (i == 0)
                return 1;
        };
    }
}
```

编译器错误 CS1644

错误消息

无法使用功能“feature”，因为它不是标准化 ISO C# 语言规范的组成部分

如果您指定了 [/langversion](#) 选项 ISO-1，而您编译的代码所使用的功能不是 ISO 1.0 标准的组成部分，则会发生此错误。若要解决此错误，请不要对任何新的 C# 编译器功能使用 ISO-1 兼容性选项。有关新的 C# 编译器功能的列表，请参见 [C# 2.0 语言和编译器中的新增功能](#)。

下面的示例生成 CS1644：

```
// CS1644.cs
// compile with: /langversion:ISO-1 /target:library
class C<T> {} // CS1644
```

编译器错误 CS1646

错误消息

关键字、标识符或字符串应位于逐字说明符 @ 之后

请参见逐字说明符“@”用法的字符串文本。逐字说明符只允许在字符串、关键字或标识符之前使用。若要解决此错误，请从任何不适当的位置移除 @ 符，或者添加所需要的字符串、关键字或标识符。

下面的示例生成 CS1646：

```
// CS1646
class C
{
    int i = @5; // CS1646
    // Try this line instead:
    // int i = 5;
}
```

编译器错误 CS1647

错误消息

表达式太长或太复杂，无法在“code”附近编译

编译器处理您的代码时出现堆栈溢出。要解决此错误，请简化您的代码。如果代码是有效的，请与[产品支持](#)联系。

编译器错误 CS1648

错误消息

无法修改只读字段“identifier”的成员(在构造函数中或在变量初始值设定项中除外)

当您试图修改一个只读字段的成员时，由于不允许修改只读字段的成员，将出现此错误。若要解决此错误，请将只读字段的赋值限定为构造函数或变量初始值设定项，或者从字段的声明中移除 `readonly` 关键字。

下面的示例生成 CS1648：

```
// CS1648.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public readonly Inner inner = new Inner();
}

class D
{
    static void Main()
    {
        Outer outer = new Outer();
        outer.inner.i = 1; // CS1648
    }
}
```

编译器错误 CS1649

错误消息

无法向只读字段“identifier”的成员传递 ref 或 out 参数(构造函数中除外)

如果将变量作为 **ref** 或 **out** 参数传递给一个属于 **readonly** 字段的成员的函数，则会发生此错误。由于该函数可能修改 **ref** 和 **out** 参数，所以这是不允许的。若要解决此错误，请移除字段上的 **readonly** 关键字，或者不要将 **readonly** 字段的成员传递给该函数。例如，您可能尝试创建一个可以被修改的临时变量，并将该临时变量作为 **ref** 参数传递，如下面的示例所示。

示例

下面的示例生成 CS1649：

```
// CS1649.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public readonly Inner inner = new Inner();
}

class D
{
    static void f(ref int iref)
    {

    }

    static void Main()
    {
        Outer outer = new Outer();
        f(ref outer.inner.i); // CS1649
        // Try this code instead:
        // int tmp = outer.inner.i;
        // f(ref tmp);
    }
}
```

编译器错误 CS1650

错误消息

无法对静态只读字段“identifier”的字段赋值(在静态构造函数或变量初始值设定项中除外)

当您试图修改只读静态字段的成员时，由于不允许修改这类成员，将出现此错误。若要解决此错误，请将只读字段的赋值限定为构造函数或变量初始值设定项，或者从字段的声明中移除 **readonly** 关键字。

```
// CS1650.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public static readonly Inner inner = new Inner();
}

class D
{
    static void Main()
    {
        Outer.inner.i = 1; // CS1650
    }
}
```

编译器错误 CS1651

错误消息

无法向静态只读字段“identifier”的字段传递 ref 或 out(静态构造函数中除外)

如果将变量作为 ref 参数传递给一个属于静态只读字段成员的函数，则会发生此错误。由于该函数可能修改 ref 参数，所以这是不允许的。若要解决此错误，请移除字段上的 **readonly** 关键字，或者不要将 readonly 字段的成员传递给该函数。例如，您可能尝试创建一个可以被修改的临时变量，并将该临时变量作为 ref 参数传递，如下面的示例所示。

下面的示例生成 CS1651：

```
// CS1651.cs
public struct Inner
{
    public int i;
}

class Outer
{
    public static readonly Inner inner = new Inner();
}

class D
{
    static void f(ref int iref)
    {
    }

    static void Main()
    {
        f(ref Outer.inner.i); // CS1651
        // Try this instead:
        // int tmp = Outer.inner.i;
        // f(ref tmp);
    }
}
```

编译器错误 CS1652

错误消息

无法修改“identifier”的成员，因为它是只读的。

当您尝试修改一个只读变量的成员时，由于它的上下文的缘故，将出现此错误。

编译器错误 CS1653

错误消息

无法将“identifier”的字段作为 ref 或 out 参数传递，因为它是只读的

如果您的变量因上下文的缘故是只读的，而您试图将此变量的成员作为 ref 或 out 参数传递，则会发生此错误。

编译器错误 CS1654

错误消息

无法修改“variable”的成员，因为它是一个“read-only variable type”

当您试图修改一个只读变量的成员时发生此错误，因为它在特殊的结构中。

示例

下面的示例生成错误 CS1654。

```
// CS1654.cs
using System;
using System.Collections;

public struct Test : IEnumerable
{
    private int index;
    public int Index
    {
        get { return index; }
        set { index = value; }
    }

    public IEnumerator GetEnumerator()
    {
        for(int i = 0; i < 10; i++)
            yield return this;
        yield break;
    }
}

public class Repro
{
    static int Main()
    {
        int i = 0;
        Test t = new Test();
        foreach (Test tt in t)
        {
            tt.Index = i++; // CS1654
        }
        return 1;
    }
}
```

编译器错误 CS1655

错误消息

无法将“variable”的字段作为 ref 或 out 参数传递，因为它是“readonly variable type”

如果您尝试将 `foreach` 变量、`using` 变量或 `fixed` 变量的成员作为 ref 或 out 参数传递给函数，则会发生此错误。因为这些变量在这些上下文中被视为只读变量，而这是不允许的。

下面的示例生成 CS1655：

```
// CS1655.cs
struct S
{
    public int i;
}

class CMain
{
    static void f(ref int iref)
    {
    }

    public static void Main()
    {
        S[] sa = new S[10];
        foreach(S s in sa)
        {
            CMain.f(ref s.i); // CS1655
        }
    }
}
```

编译器错误 CS1656

错误消息

无法给“variable”赋值，因为它是“read-only variable type”

当变量赋值发生在只读上下文中时，将出现此错误。只读上下文包括 `foreach` 迭代变量、`using` 变量和 `fixed` 变量。若要解决此错误，请避免在 `using` 块、`foreach` 语句和 `fixed` 语句中给变量赋值。

示例

下面的示例生成 CS1656：

```
// CS1656.cs
// compile with: /unsafe
using System;

class C : IDisposable
{
    public void Dispose() { }
}

class CMain
{
    unsafe public static void Main()
    {
        using (C c = new C())
        {
            c = new C(); // CS1656
        }

        foreach (object o in new string[] { "1", "2" })
        {
            o = "10"; // CS1656
        }

        int[] ary = new int[] { 1, 2, 3, 4 };
        fixed (int* p = ary)
        {
            p = null; // CS1656
        }
    }
}
```

编译器错误 CS1657

错误消息

无法将“variable”作为 ref 或 out 参数传递，因为它是一个“readonly variable type”

当一个变量在该变量为只读的上下文中作为 `ref` 或 `out` 参数传递时，将出现此错误。只读上下文包括 `foreach` 迭代变量、`using` 变量和 `fixed` 变量。若要解决此错误，请不要调用在 `using` 块、`foreach` 语句和 `fixed` 语句中接受 `foreach`、`using` 或 `fixed` 变量作为 `ref` 或 `out` 参数的函数。

示例

下面的示例生成 CS1657：

```
// CS1657.cs
using System;
class C : IDisposable
{
    public int i;
    public void Dispose() {}
}

class CMain
{
    static void f(ref C c)
    {
    }
    static void Main()
    {
        using (C c = new C())
        {
            f(ref c); // CS1657
        }
    }
}
```

下面的代码说明 `fixed` 语句中的同样的问题：

```
// CS1657b.cs
// compile with: /unsafe
unsafe class C
{
    static void F(ref int* p)
    {
    }

    static void Main()
    {
        int[] a = new int[5];
        fixed(int* p = a) F(ref p); // CS1657
    }
}
```

编译器错误 CS1660

错误消息

无法将匿名方法块转换为类型“type”，因为它不是一个委托类型

如果您尝试将匿名方法块分配给或以其他方式转换为不是委托类型的类型，就会发生此错误。

下面的示例生成 CS1660：

```
// CS1660.cs
delegate int MyDelegate();
class C {
    static void Main()
    {
        int i = delegate { return 1; }; // CS1660
        // Try this instead:
        // MyDelegate myDelegate = delegate { return 1; };
        // int i = myDelegate();
    }
}
```

编译器错误 CS1661

错误消息

无法将匿名方法块转换为委托类型“delegate type”，因为指定块的参数类型和委托参数类型不匹配

如果在匿名方法定义中，匿名方法的参数类型和委托参数类型不匹配，则将出现此错误。请检查参数数目、参数类型和所有 ref 或 out 参数，验证是否完全匹配。

下面的示例生成 CS1661：

```
// CS1661.cs

delegate void MyDelegate(int i);

class C
{
    public static void Main()
    {
        MyDelegate d = delegate(string s) { }; // CS1661
    }
}
```

编译器错误 CS1662

错误消息

无法将匿名方法块转换为委托类型“delegate type”，原因是块中的某些返回类型无法隐式转换为委托返回类型

如果匿名方法块的返回语句有一个无法隐式转换为委托的返回类型的类型，则会发生此错误。

下面的示例生成 CS1662：

```
// CS1662.cs

delegate int MyDelegate(int i);

class C
{
    public static void Main()
    {
        MyDelegate d = delegate(int i) { return 1.0; }; // CS1662
        // Try this instead:
        // MyDelegate d = dekegate(int i) { return (int)1.0; };
    }
}
```

编译器错误 CS1663

错误消息

固定大小的缓冲区类型必须为下列类型之一: bool、byte、short、int、long、char、sbyte、ushort、uint、ulong、float 或 double

固定大小的缓冲区不能是上面列出的类型以外的任何其他类型。若要避免此错误, 请使用其他类型或者不使用固定数组。

示例

下面的示例生成 CS1663。

```
// CS1663.cs
// compile with: /unsafe /target:library

unsafe struct C
{
    fixed string ab[10];    // CS1663
}
```

编译器错误 CS1664

错误消息

长度为"length"且类型为"type"的固定大小缓冲区太大

固定大小缓冲区的最大大小(由长度乘以元素大小决定)为 $2^{31} = 268435455$ 。

编译器错误 CS1665

错误消息

固定大小的缓冲区的长度必须大于零

如果固定大小的缓冲区的大小被声明为零或负值，则发生此错误。固定大小的缓冲区的长度必须是正整数。

示例

下面的示例生成 CS1665。

```
// CS1665.cs
// compile with: /unsafe /target:library
struct S
{
    public unsafe fixed int A[0];    // CS1665
}
```

编译器错误 CS1666

错误消息

不能使用非固定表达式中包含的固定大小缓冲区。请尝试使用 `fixed` 语句。

如果您在一个涉及本身不固定的类的表达式中使用固定大小缓冲区，则会出现此错误。运行库可以自由地将未固定的类到处移动以优化内存访问，这在使用固定大小缓冲区时可能会导致错误。为避免此错误，请在语句中使用 `fixed` 关键字。

示例

下面的示例生成 CS1666。

```
// CS1666.cs
// compile with: /unsafe /target:library
unsafe struct S
{
    public fixed int buffer[1];
}

unsafe class Test
{
    S field = new S();

    private bool example1()
    {
        return (field.buffer[0] == 0);      // CS1666 error
    }

    private bool example2()
    {
        // OK
        fixed (S* p = &field)
        {
            return (p->buffer[0] == 0);
        }
    }

    private bool example3()
    {
        S local = new S();
        return (local.buffer[0] == 0);
    }
}
```

编译器错误 CS1667

错误消息

属性 (Attribute)"attribute"在属性 (Property) 或事件访问器上无效。它仅在"declaration type"声明上有效。

如果您在属性 (Property) 或事件访问器上使用属性 (Attribute), 而属性 (Attribute) 应在属性 (Property) 或事件本身上时, 会发生此错误。属性 (Attribute) [CLSCompliantAttribute](#)、[ConditionalAttribute](#) 和 [ObsoleteAttribute](#) 会发生此错误。

示例

下面的示例生成 CS1670:

```
// CS1667.cs
using System;

public class C
{
    private int i;

    //Try this instead:
    //#[Obsolete]
    public int ObsoleteProperty
    {
        [Obsolete] // CS1667
        get { return i; }
        set { i = value; }
    }

    public static void Main()
    {
    }
}
```

编译器错误 CS1670

错误消息

params 在此上下文中无效

一些 C# 功能与变量参数列表不兼容，并且不允许使用 **params** 关键字，包括：

- 匿名方法的参数列表
- 重载运算符

示例

下面的示例生成 CS1670：

```
// CS1670.cs
public class C
{
    public bool operator +(params int[] paramsList) // CS1670
    {
        return false;
    }

    static void Main()
    {
    }
}
```

编译器错误 CS1671

错误消息

命名空间声明不能有修饰符或属性

修饰符在应用于命名空间时没有意义，因此不允许使用修饰符。

下面的示例生成 CS1671：

```
// CS1671.cs
public namespace NS // CS1671
{
}
```

编译器错误 CS1672

错误消息

选项“option”对 /platform 无效; 必须是 anycpu、x86、Itanium 或 x64

这些选项指定处理器类型; 使用其中一个列出的形式。

编译器错误 CS1673

错误消息

结构内部的匿名方法无法访问“this”的实例成员。请考虑将“this”复制到匿名方法外部的某个局部变量并改用该局部变量

下面的示例生成 CS1673:

```
// CS1673.cs
delegate int MyDelegate();

public struct S
{
    int member;

    public int F(int i)
    {
        member = i;
        // Try assigning to a local variable
        // S s = this;
        MyDelegate d = delegate()
        {
            i = this.member; // CS1673
            // And use the local variable instead of "this"
            // i = s.member;
            return i;
        };
        return d();
    }
}

class CMain
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1674

错误消息

"T":using 语句中使用的类型必须可以隐式转换为"System.IDisposable"

using 语句 旨在用于确保在 **using** 块的最后处置对象，因此只有可处置的类型才能在这类语句中使用。例如，值类型不是可处置的，并且没有约束为类的类型参数不能假定为可处置。

示例

下面的示例生成 CS1674。

```
// CS1674.cs
class C
{
    public static void Main()
    {
        int a = 0;
        a++;

        using (a) {} // CS1674
    }
}
```

下面的示例生成 CS1674。

```
// CS1674_b.cs
using System;
class C {
    public void Test() {
        using (C c = new C()) {} // CS1674
    }
}

// OK
class D : IDisposable {
    void IDisposable.Dispose() {}
    public void Dispose() {}

    public static void Main() {
        using (D d = new D()) {}
    }
}
```

下面的示例说明了需要使用一个类类型约束来确保未知的类型参数是可释放的。下面的示例生成 CS1674。

```
// CS1674_c.cs
// compile with: /target:library
using System;
public class C<T>
// Add a class type constraint that specifies a disposable class.
// Uncomment the following line to resolve.
// public class C<T> where T : IDisposable
{
    public void F(T t)
    {
        using (t) {} // CS1674
    }
}
```

编译器错误 CS1675

错误消息

枚举不能有类型参数

若要解决此错误, 请从 **enum** 声明中移除类型参数。

示例

下面的示例生成 CS1675:

```
// CS1675.cs
enum E<T> // CS1675
{
}

class CMain
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1676

错误消息

参数“number”必须用“keyword”关键字声明

当匿名方法中的参数类型修饰符与您要将该方法强制转换为的委托的声明中使用的修饰符不同时，会出现此错误。

下面的示例生成 CS1676：

```
// CS1676.cs
delegate void E(ref int i);
class Errors
{
    static void Main()
    {
        E e = delegate(out int i) { };    // CS1676
        // To resolve, use the following line instead:
        // E e = delegate(ref int i) { };
    }
}
```

编译器错误 CS1677

错误消息

参数“number”不应用“keyword”关键字声明

当匿名方法中的参数类型修饰符与您要将该方法强制转换为的委托的声明中使用的修饰符不匹配时，则会发生此错误。

示例

下面的示例生成 CS1677:

```
// CS1677.cs
delegate void D(int i);
class Errors
{
    static void Main()
    {
        D d = delegate(out int i) { };    // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int i) { };

        D d = delegate(ref int j){}; // CS1677
        // To resolve, use the following line instead:
        // D d = delegate(int j){};
    }
}
```

编译器错误 CS1678

错误消息

参数“number”被声明为类型“type1”，但应为“type2”

当匿名方法中的参数类型与您要将方法强制转换为的委托的声明不同时，将出现此错误。

下面的示例生成 CS1678：

```
// CS1678
delegate void D(int i);
class Errors
{
    static void Main()
    {
        D d = delegate(string s) { };    // CS1678
        // To resolve, use the following line instead:
        // D d = delegate(int s) { };
    }
}
```

编译器错误 CS1679

错误消息

"/reference"的外部别名无效;"identifier"不是有效的标识符

当使用 **/reference** 选项的外部程序集别名功能时, 根据 C# 语言规范, 在 **/reference:** 之后以及在"="之前的文本必须是一个有效的 C# 标识符或关键字。

要更正此错误, 请将"="之前的文本更改为有效的 C# 标识符或关键字。

示例

下面的示例生成 CS1679。

```
// CS1679.cs
// compile with: /reference:123$BadIdentifier%=System.dll
class TestClass {
    static void Main()
    {
    }
}
```

编译器错误 CS1680

错误消息

无效的引用别名选项：“alias=”-- 缺少文件名。

如果您在使用 **/reference** 编译器选项的 `alias` 功能时没有指定有效的文件名，则会发生此错误。

下面的示例生成 CS1680。

```
// CS1680.cs
// compile with: /reference:alias=
// CS1680 expected
// To resolve, specify the name of a file with an assembly manifest
class MyClass {}
```

编译器错误 CS1681

错误消息

不能重新定义全局外部别名

全局别名已被定义为包括所有的非别名引用, 因此不能重新定义。

示例

下面的示例生成 CS1681。

```
// CS1681.cs
// compile with: /reference:global=System.dll
// CS1681 expected

// try this instead: /reference:System.dll
class A
{
    static void Main() {}
}
```

编译器错误 CS1686

错误消息

不能在匿名方法块内部获取并使用局部变量“variable”或其成员的地址

当您使用某个变量并尝试获取其地址时，只要这些操作有一项是在匿名方法内部执行的，就会生成此错误。

示例

下面的示例生成 CS1686。

```
// CS1686.cs
// compile with: /unsafe /target:library
class MyClass
{
    public unsafe delegate int * MyDelegate();

    public unsafe int * Test()
    {
        int j = 0;
        MyDelegate d = delegate { return &j; };      // CS1686
        return &j;      // OK
    }
}
```

编译器错误 CS1688

错误消息

无法将没有参数列表的匿名方法块转换为委托类型“delegate”，因为它有一个或多个 out 参数。

编译器允许在大多数情况下从匿名方法块中省略参数。当匿名方法块没有参数列表，而委托具有 out 参数时，会发生此错误。编译器不允许这种情况，因为它需要忽略 out 参数的存在，而这不可能是正确的行为。

示例

下面的代码生成错误 CS1688。

```
// CS1688.cs
using System;
delegate void OutParam(out int i);
class ErrorCS1676
{
    static void Main()
    {
        OutParam o;
        o = delegate // CS1688
        // Try this instead:
        // o = delegate(out int i)
        {
            Console.WriteLine("");
        };
    }
}
```

编译器错误 CS1689

错误消息

属性“Attribute Name”仅对方法或属性类有效

只有 **ConditionalAttribute** 属性会出现此错误。正如错误信息中所指出的，此属性只能在方法或属性类上使用。例如，试图将此属性应用于类会产生此错误。

示例

下面的示例生成 CS1689。

```
// CS1689.cs
// compile with: /target:library
[System.Diagnostics.Conditional("A")]    // CS1689
class MyClass {}
```

编译器警告(等级 1)CS1690

错误消息

由于“member”是引用封送类的字段，访问上面的成员可能导致运行时异常

当您尝试对从 [MarshalByRefObject](#) 派生的类成员调用方法、属性或索引器时，并且该成员是值类型时，会出现此警告。若要解决此警告，请将该成员复制到局部变量并对此变量调用方法。

下面的示例生成 CS1690：

```
// CS1690.cs
using System;

class WarningCS1690: MarshalByRefObject
{
    int i = 5;

    public static void Main()
    {
        WarningCS1690 e = new WarningCS1690();
        e.i.ToString();    // CS1690

        // OK
        int i = e.i;
        i.ToString();
        e.i = i;
    }
}
```

编译器错误 CS1703

错误消息

已导入具有相同标识“Assembly Name”的程序集。尝试移除其中的一个重复引用。

编译器移除有相同的路径和文件名称的引用，但有可能相同的文件在两个位置都存在，或者您忘了更改版本号。此错误指出两个引用具有相同的程序集标识，因此编译器无法在元数据中区分它们。或者移除其中的一个冗余引用，或者用某种办法使引用一致，例如通过提高程序集版本号。

以下代码生成错误 CS1703。

示例

此代码在目录 .\bin1 中创建程序集 A。

将此示例保存在名为 CS1703a1.cs 的文件中，并用以下标志编译该文件:
/t:library /out:.\\bin1\\cs1703.dll
/keyfile:key.snk

```
using System;
public class A { }
```

此代码在目录 .\bin2 中创建程序集 A 的副本。

将此示例保存在名为 CS1703a2.cs 的文件中，并用以下标志编译该文件:
/t:library /out:.\\bin2\\cs1703.dll
/keyfile:key.snk

```
using System;
public class A { }
```

此代码引用前面两个模块中的程序集 A。

将此示例保存在名为 CS1703ref.cs 的文件中，并用以下标志编译该文件:
/t:library /r:A2=.\\bin2\\cs1703.dll
/r:A1=.\\bin1\\cs1703.dll

```
extern alias A1;
extern alias A2;
```

编译器错误 CS1704

错误消息

已导入具有相同简单名称“Assembly Name”的程序集。请尝试移除其中一个引用，或者给引用加上签名以启用并列模式。

此错误指出有两个引用具有相同的程序集标识，这是因为相关程序集缺少强名称而且没有签名，因而编译器无法在元数据中区分它们。因此，运行时会忽略版本和区域性程序集名称属性。用户应移除多余的引用，重命名其中一个引用，或者为它们提供一个强名称。

示例

此示例创建一个程序集并将它保存到根目录下。

```
// CS1704_a.cs
// compile with: /target:library /out:c:\\cs1704.dll
public class A {}
```

此示例创建一个与上面的示例同名的程序集，但将它保存到不同的位置。

```
// CS1704_b.cs
// compile with: /target:library /out:cs1704.dll
public class A {}
```

此示例试图引用这两个程序集。下面的示例生成 CS1704。

```
// CS1704_c.cs
// compile with: /target:library /r:A2=cs1704.dll /r:A1=c:\\cs1704.dll
// CS1704 expected
extern alias A1;
extern alias A2;
```

编译器错误 CS1705

错误消息

程序集“AssemblyName1”所使用的“TypeName”的版本高于所引用的程序集“AssemblyName2”的版本

您要引用一个版本号高于所引用的程序集的版本号的类型。

例如，您有两个程序集：A 和 B。A 引用的类 myClass 被添加到 2.0 版的程序集 B 中。但是，对程序集 B 的引用指定的是 1.0 版。编译器有一致性绑定引用规则，对 2.0 版的引用无法由 1.0 版满足。

示例

本示例由 4 个代码模块组成：

- 两个 DLL，它们除了版本属性外，其他方面都是一样的。
- 一个引用它们的 DLL。
- 客户端。

下面是两个相同的 DLL 中的第一个。

```
// CS1705_a.cs
// compile with: /target:library /out:c:\\cs1705.dll /keyfile:mykey.snk
[assembly:System.Reflection.AssemblyVersion("1.0")]
public class A
{
    public void M1() {}
    public class N1 {}
    public void M2() {}
    public class N2 {}
}

public class C1 {}
public class C2 {}
```

下面是程序集 2.0 版，由属性 [AssemblyVersionAttribute](#) 指定。

```
// CS1705_b.cs
// compile with: /target:library /out:cs1705.dll /keyfile:mykey.snk
using System.Reflection;
[assembly: AssemblyVersion("2.0")]
public class A
{
    public void M2() {}
    public class N2 {}
    public void M1() {}
    public class N1 {}
}

public class C2 {}
public class C1 {}
```

将此示例保存在一个名为 CS1705ref.cs 的文件中，并使用以下标志编译该文件：/t:library /r:A2=.\\bin2\\CS1705a.dll /r:A1=.\\bin1\\CS1705a.dll

```
// CS1705_c.cs
// compile with: /target:library /r:A2=c:\\CS1705.dll /r:A1=CS1705.dll
extern alias A1;
extern alias A2;
using a1 = A1::A;
using a2 = A2::A;
using n1 = A1::A.N1;
using n2 = A2::A.N2;
```

```
public class Ref
{
    public static a1 A1() { return new a1(); }
    public static a2 A2() { return new a2(); }
    public static A1::C1 M1() { return new A1::C1(); }
    public static A2::C2 M2() { return new A2::C2(); }
    public static n1 N1() { return new a1.N1(); }
    public static n2 N2() { return new a2.N2(); }
}
```

下面的示例引用 1.0 版的 CS1705.dll 程序集。但语句 `Ref.A2().M2()` 引用 CS1705_c.dll 的类中的 A2 方法，它将返回 a2(别名为 A2::A)，而 A2 通过 **extern** 语句引用 2.0 版，因此导致版本不匹配。

下面的示例生成 CS1705。

```
// CS1705_d.cs
// compile with: /reference:c:\\CS1705.dll /reference:CS1705_c.dll
// CS1705 expected
class Tester
{
    static void Main()
    {
        Ref.A1().M1();
        Ref.A2().M2();
    }
}
```

编译器错误 CS1706

错误消息

表达式不能包含匿名方法

不能在表达式内插入匿名方法。

更正此错误

- 在表达式中使用规则 **delegate**。

示例

下面的示例生成 CS1706。

```
// CS1706.cs
using System;

delegate void MyDelegate();
class MyAttribute : Attribute
{
    public MyAttribute(MyDelegate d) { }
}

// Anonymous Method in Attribute declaration is not allowed.
[MyAttribute(delegate{/* anonymous Method in Attribute declaration */})] // CS1706
class Program
{
}
```

编译器错误 CS1708

错误消息

固定大小的缓冲区只能通过局部变量或字段访问

C# 2.0 的一个新功能是能够在 **struct** 的内部定义内嵌数组。这类数组只能通过局部变量或字段访问，不能作为表达式左侧的中间值被引用。此外，这些数组无法通过 **static** 或 **readonly** 的字段访问。

要解决此错误，请定义一个数组变量，然后将内嵌数组分配给该变量。或者，从表示内嵌数组的字段中移除 **static** 或 **readonly** 修饰符。

示例

下面的示例生成 CS1708。

```
// CS1708.cs
// compile with: /unsafe
using System;

unsafe public struct Foo
{
    public fixed char name[10];
}

public unsafe class C
{
    public Foo UnsafeMethod()
    {
        Foo myFoo = new Foo();
        return myFoo;
    }

    static void Main()
    {
        C myC = new C();
        myC.UnsafeMethod().name[3] = 'a'; // CS1708
        // Uncomment the following 2 lines to resolve:
        // Foo myFoo = myC.UnsafeMethod();
        // myFoo.name[3] = 'a';

        // The field cannot be static.
        C._foo1.name[3] = 'a'; // CS1708

        // The field cannot be readonly.
        myC._foo2.name[3] = 'a'; // CS1708
    }

    static readonly Foo _foo1;
    public readonly Foo _foo2;
}
```

编译器错误 CS1713

错误消息

生成类型 Typename1—'Reason' 的元数据名称时发生意外错误。

此错误通常是由内部编译器错误导致的。对您的代码稍稍进行更改，如缩短名称的长度，然后重新编译，便可能解决此问题。

编译器错误 CS1714

错误消息

TypeName1 的基类或接口未能解析或无效

您要从某个类或接口实现 TypeName1，而该类或接口无法被解析(例如，编译器无法找到它)或者无效。解决方法是确定它是这两种情况中的哪一种，然后要么更准确地指定类型的位置，要么修复基类中的任何编译器错误。

编译器错误 CS1715

错误消息

"Type1":类型必须为"Type2"才能与重写成员"MemberName"匹配

此错误与[编译器错误 CS0508](#)相同，只不过CS0508现在仅适用于具有返回类型的方法，而CS1715适用于仅具有“types”而不是“return types”的属性和索引器。

示例

下面的代码生成CS1715。

```
// CS1715.cs
abstract public class Base
{
    abstract public int myProperty
    {
        get;
        set;
    }
}

public class Derived : Base
{
    int myField;
    public override double myProperty // CS1715
    // try the following line instead
    // public override int myProperty
    {
        get { return myField; }
        set { myField+= value; }
    }

    public static void Main()
    {
        Derived d = new Derived();
        d.myProperty = 5;
    }
}
```

编译器错误 CS1716

错误消息

不要使用“System.Runtime.CompilerServices.FixedBuffer”属性。请改用“fixed”字段修饰符。

在包含与字段声明相似的固定大小的数组声明的不安全代码段中，会发生此错误。不要使用此属性。请改用关键字 **fixed**。

示例

下面的示例生成 CS1716。

```
// CS1716.cs
// compile with: /unsafe
using System;
using System.Runtime.CompilerServices;

public struct UnsafeStruct
{
    [FixedBuffer(typeof(int), 4)] // CS1716
    unsafe public int aField;
    // Use this single line instead of the above two lines.
    // unsafe public fixed int aField[4];
}

public class TestUnsafe
{
    static int Main()
    {
        UnsafeStruct us = new UnsafeStruct();
        unsafe
        {
            if (us.aField[0] == 0)
                return us.aField[1];
            else
                return us.aField[2];
        }
    }
}
```

编译器错误 CS1719

错误消息

读取 Win32 资源文件“File Name”时出错 --“reason”

读取 Win32 资源文件的尝试失败，原因已在错误中给出，通常的原因是“未找到文件”或“访问被拒绝”之类。按照原因的描述更正问题可解决此错误。

编译器错误 CS1721

错误消息

类“class”不能具有多个基类: “class_1”和“class_2”

导致此错误的最常见原因是试图使用多重继承。C# 中的类只能从一个类继承: 类声明中位于类名称后面的其余的类都必须是接口。

示例

下面的示例生成 CS1721。

```
// CS1721.cs
public class A {}
public class B {}
public class MyClass : A, B {} // CS1721
```

编译器错误 CS1722

错误消息

基类“class”必须在任何接口之前

指定从中继承的类和要实现的接口时，必须首先指定类名。

示例

下面的示例生成 CS1722。

```
// CS1722.cs
// compile with: /target:library
public class A {}
interface I {}

public class MyClass : I, A {}    // CS1722
public class MyClass2 : A, I {}    // OK
```

编译器错误 CS1724

错误消息

为“System.Runtime.InteropServices.DefaultCharsetAttribute”的参数指定的值无效

此错误由 [DefaultCharsetAttribute](#) 类的无效参数生成。

示例

下面的示例生成 CS1724。

```
// CS1724.cs
using System.Runtime.InteropServices;
// To resolve, replace 42 with a valid CharSet value.
[module:DefaultCharsetAttribute((CharSet)42)]    // CS1724
class C {

[DllImport("F.Dll")]
extern static void FW1Named();

static void Main() {}
}
```

编译器错误 CS1728

错误消息

无法将委托绑定到作为“type”成员的“member”

无法将委托绑定到 **Nullable** 类型的成员。

示例

下面的示例生成 CS1728。

```
// CS1728.cs
// compile with: /W:2
class Test
{
    delegate T GetT<T>();
    delegate T GetT1<T>(T t);

    delegate bool E(object o);
    delegate int I();
    delegate string S();

    static void Main()
    {
        int? x = null;
        int? y = 5;

        GetT<int> d1 = x.GetValueOrDefault;    // CS1728
        GetT<int> d2 = y.GetValueOrDefault;    // CS1728
        GetT1<int> d3 = x.GetValueOrDefault;   // CS1728
        GetT1<int> d4 = y.GetValueOrDefault;   // CS1728
    }
}
```

编译器错误 CS1900

错误消息

警告等级必须在 0-4 的范围内

[/warn](#) 编译器选项只能带五个可能的值(0、1、2、3 或 4)之一。传递给 **/warn** 的任何其他值均将导致 CS1900。

下面的示例生成 CS1900:

```
// CS1900.cs
// compile with: /W:5
// CS1900 expected
class x
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1902

错误消息

用于 /debug 的选项“option”无效; 选项必须是 full 或 pdbonly

给 [/debug](#) 编译器选项传递的选项无效。

下面的示例生成 CS1902:

```
// CS1902.cs
// compile with: /debug:x
// CS1902 expected
class x
{
    public static void Main()
    {
    }
}
```

编译器错误 CS1906

错误消息

选项“option”无效; 资源可见性必须是“public”或“private”

此错误指示一个无效的 [/resource\(将资源文件嵌入到输出中\)](#) 或 [/linkresource\(链接到 .NET Framework 资源\)](#) 命令行选项。请检查 **/resource** 或 **/linkresource** 命令行选项的语法, 确保所使用的可访问性修饰符是 **public** 或 **private**。

编译器错误 CS1908

错误消息

DefaultValue 属性的变量类型必须与参数类型匹配

对 [DefaultValueAttribute](#) 属性值使用错误的参数时生成此错误。请使用与参数类型匹配的值。

示例

下面的示例生成 CS1908。

```
// CS1908.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
    void Bad([Optional] [DefaultParameterValue("true")] bool b);    // CS1908
    void Good([Optional] [DefaultParameterValue(true)] bool b);    // OK
}
```

编译器错误 CS1909

错误消息

DefaultValue 属性不适用于“type”类型的参数

使用不适用于此参数类型的 DefaultValue 属性时，会生成 CS1909。

示例

下面的示例生成 CS1909。

```
// CS1909.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface ISomeInterface
{
    void Test1([DefaultParameterValue(new int[] {1, 2})] int[] arr1);    // CS1909
    void Test2([DefaultParameterValue("Test String")] string s);    // OK
}
```

编译器错误 CS1910

错误消息

"type"类型的参数不适用于 DefaultValue 属性

对于类型为对象的参数, [DefaultParameterValueAttribute](#) 的变量必须是 **null**、整型、浮点型、**bool**、**string**、**enum** 或 **char**。该变量不能是 [Type](#) 类型或任何数组类型。

示例

下面的示例生成 CS1910。

```
// CS1910.cs
// compile with: /target:library
using System.Runtime.InteropServices;

public interface MyI
{
    void Test([DefaultValue(typeof(object))] object o);    // CS1910
}
```

编译器错误 CS2000

错误消息

编译器初始化意外失败

此错误指示初始化失败。

使用安装程序来修复或重新安装 Visual Studio 或 .NET Framework SDK。

如果错误仍然存在, 请与产品支持部门联系。

编译器错误 CS2001

错误消息

未能找到源文件“file”

给编译器传递了源文件名，但未能找到该文件。请检查文件名的拼写和文件的位置。

编译器错误 CS2003

错误消息

响应文件“file”被包含多次

响应文件多次传递给了编译器。对于每个输出文件，响应文件只能传递给编译器一次。

有关响应文件的详细信息，请参见 [@\(指定响应文件\)](#)。

编译器错误 CS2005

错误消息

"option"选项缺少文件规范

只部分指定了[编译器选项](#)。

例如, 当使用 [/recurse](#) 时, 必须指定要搜索的文件:[/recurse:filename.cs](#)。

示例

下面的示例生成 CS2005。

```
// CS2005.cs
// compile with: /recurse:
// CS2005 expected
class x
{
    public static void Main() {}
}
```

编译器错误 CS2006

错误消息

命令行语法错误:"option"选项缺少"text"

option 的语法要求附加的文本。有关信息, 请参见[编译器选项](#)。

编译器错误 CS2007

错误消息

无法识别的命令行选项：“option”

传递给编译器的字符串不是[编译器选项](#)，即使它以正斜杠 (/) 开头。

下面的示例生成 CS2007：

```
// CS2007.cs
// compile with: /recur
// CS2007 expected
class x
{
    public static void Main() {}
```

编译器错误 CS2008

错误消息

未指定输入

调用了编译器并指定了编译器选项，但未传递任何源代码文件。

编译器错误 CS2011

错误消息

无法打开响应文件“file”

在编译中指定了响应文件，但编译器未能找到并打开该文件。

有关响应文件的详细信息，请参见 [@\(指定响应文件\)](#)。

编译器错误 CS2012

错误消息

无法打开“file”进行写入

当使用 [`/bugreport:file`](#) 编译器选项时，未能打开文件进行写入。确保指定的文件名有效并且该文件不是只读的。

编译器错误 CS2013

错误消息

无效的图像基数“value”

传递给 [/baseaddress](#) 编译器选项的值无效(不是数字)。

下面的示例生成 CS2013:

```
// CS2013.cs
// compile with: /target:library /baseaddress:x
// CS2013 expected
class MyClass
{
}
```

编译器错误 CS2015

错误消息

"file"是二进制文件而不是文本文件

传递给编译器的文件是二进制文件。而编译器需要的是源代码文件。

编译器错误 CS2016

错误消息

代码页“codepage”无效或未安装

传递给 [/codepage](#) 编译器选项的值无效。

下面的示例生成 CS2016:

```
// CS2016.cs
// compile with: /codepage:x
// CS2016 expected
class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS2017

错误消息

如果生成模块或库，则无法指定 /main

当正在生成 [/target:library](#) 时不能指定 Main 入口点。

下面的示例生成 CS2017:

```
// CS2017.cs
// compile with: /main:MyClass /target:library
// CS2017 expected
class MyClass
{
    public static void Main()
    {
    }
}
```

编译器错误 CS2018

错误消息

无法找到消息文件“cscmsgs.dll”

未找到包含编译器的错误信息和警告消息的 .dll 文件。该文件与其他编译器支持文件必须在同一目录中。

编译器错误 CS2019

错误消息

/target 的目标类型无效: 必须指定“exe”、“winexe”、“library”或“module”

使用了 [/target](#) 编译器选项, 但传递了无效参数。若要解决该错误, 请使用适合输出文件的 **/target** 选项格式重新编译程序。

下面的示例生成 CS2017:

```
// CS2019.cs
// compile with: /target:libra
// CS2019 expected
class MyClass
{}
```

编译器错误 CS2020

错误消息

只有第一组输入文件能生成非“模块”的目标

在多输出编译中, 第一个输出文件必须用 [/target:exe](#)、[/target:winexe](#) 或 [/target:library](#) 生成。后面的任何输出文件必须用 [/target:module](#) 生成。

编译器错误 CS2021

错误消息

文件名“file”太长或无效

传递给 C# 编译器的所有文件名的长度都必须小于 **_MAX_PATH**(定义在 Windows 头文件中), 该编译器将在以下情况中给出该错误:

- 文件名(包括路径)的长度大于 **_MAX_PATH**。
- 文件名包含无效字符。
- 文件名包含某些通配符, 而在此处(如资源文件名中)并不允许使用通配符。

编译器错误 CS2022

错误消息

选项“/out”和“/target”必须出现在源文件名之前

当在命令行中指定 [/out\(设置输出文件名\)](#) 和 [/target\(指定输出文件格式\)](#) 编译器选项时，它们必须位于源代码文件之前。

编译器错误 CS2024

错误消息

文件节的对齐编号“#”无效

给 [/filealign](#) 编译器选项传递的值无效。

下面的示例生成 CS2024:

```
// CS2024.cs
// compile with: /filealign:ex
// CS2024 expected
class MyClass
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS2029

错误消息

"/define"的值无效;"identifier"不是有效的标识符

如果 [/define](#) 选项中使用的值包含一些无效的字符，则将出现此警告。

编译器错误 CS2032

错误消息

命令行中或响应文件中不允许有字符“character”。

csc.exe 的响应文件和命令行选项不允许包含 0-31 范围内的低 ASCII 码控制字符或者管道 (|) 字符。您通常无法从命令行中直接生成此错误，因为命令行处理器将筛选出无效的字符，这一点与集成开发环境 (IDE) 一样。您可以使用响应文件生成此错误，如下所示：

生成此错误

1. 在“我的文档”中创建一个名为 CS2032.rsp 的文件，它包含行 /target:exe /out:cs2032.exe cs2032.cs。
2. 创建一个名为 cs2032.cs 的文件，它包含“我的文档”中的一切。
3. 单击“开始”，然后指向“所有程序”，指向“Microsoft Visual Studio 2005”，指向“Visual Studio 2005 工具”，然后单击“Visual Studio 命令提示”。

“Visual Studio 命令提示”窗口打开。

4. 在“Visual Studio 命令提示”窗口中，将当前的目录更改为“我的文档”。
5. 从“Visual Studio 命令提示”中运行以下内容:csc @cs2032.rsp
6. 出现 CS2032 错误信息。
- 7.
- 1.

编译器错误 CS2033

错误消息

包含短文件名“filename”的长文件名已存在，无法创建同名短文件名

编译一个名称长度超过八个字符的 C# 文件。然后编译另一个文件，但其名称比上一个文件短，例如在名称的前六个字符后面加上“~1”。第二个编译将生成此错误。

要解决此错误，请将短文件名重命名为一个和长文件名不冲突的名称。

编译器错误 CS2034

错误消息

一个声明外部别名的 /reference 选项只能有一个文件名。若要指定多个别名或文件名, 请使用多个 /reference 选项。

若要指定两个别名和/或文件名, 请使用两个 **/reference** 选项, 如下所示:

示例

下面的代码将生成错误 CS2034。

```
// CS2034.cs
// compile with: /r:A1=cs2034a1.dll;A2=cs2034a2.dll
// to fix, compile with: /r:A1=cs2034a1.dll /r:A2=cs2034a2.dll
// CS2034
extern alias A1;
extern alias A2;
using System;
```

编译器错误 CS2035

错误消息

命令行语法错误：“compiler_option”选项缺少“<number>”

有些编译器选项需要一个值。

示例

下面的示例生成 CS2035。

```
// CS2035.cs
// compile with: /baseaddress
// CS2035 expected
```

编译器警告(等级 1)CS3027

错误消息

"type_1"不符合 CLS, 因为基接口"type_2"不符合 CLS

只有符合 CLS 的类型才能作为符合 CLS 的类型的基类型。

示例

下面的示例包含一个接口, 此接口具有一个方法, 可在其签名中使用不符合 CLS 的类型, 从而使其类型不符合 CLS。

```
// CS3027.cs
// compile with: /target:library
public interface IBase
{
    void IMethod(uint i);
}
```

下面的示例生成 CS3027。

```
// CS3027_b.cs
// compile with: /reference:CS3027.dll /target:library /W:1
[assembly:System.CLSCompliant(true)]
public interface IDerived : IBase {}
```

编译器错误 CS5001

错误消息

程序“program”不包含适合入口点的静态“Main”方法

在生成可执行文件的代码中没有找到 [Main](#) 方法时会发生此错误。如果用错误的大小写定义入口点函数 **Main**(如小写的 **main**), 或者没有将 **Main** 声明为静态的, 也可能发生此错误。

示例

下面的示例生成 CS5001。

```
// CS5001.cs
// CS5001 expected
public class a
{
    // Uncomment the following line to resolve.
    // public static void Main() {}
}
```

编译器错误 CS0025

错误消息

未能找到标准库文件“file”

未找到编译器所需的文件。请检查路径是否正确以及文件是否存在。

如果文件是 Visual Studio 系统文件，您可能需要修复 Visual Studio 安装，或者完全重新安装它。

编译器警告(等级 1)CS0183

错误消息

给定表达式始终为所提供的("type")类型

如果条件语句始终计算为 **true**, 则不需要条件语句。当您试图使用 **is** 运算符计算类型时, 会出现此警告。如果计算的是值类型, 则检查是不必要的。

下面的示例生成 CS0183:

```
// CS0183.cs
// compile with: /W:1
using System;
public class Test
{
    public static void F(Int32 i32, String str)
    {
        if (str is Object)          // OK
            Console.WriteLine( "str is an object" );
        else
            Console.WriteLine( "str is not an object" );

        if (i32 is Object)    // CS0183
            Console.WriteLine( "i32 is an object" );
        else
            Console.WriteLine( "i32 is not an object" ); // never reached
    }

    public static void Main()
    {
        F(0, "CS0183");
        F(120, null);
    }
}
```

编译器警告(等级 1)CS0184

错误消息

给定表达式始终不是所提供的 ("type") 类型

表达式决不可能为 **true**, 因为正测试的变量既没有声明为 **type**, 也不是从 **type** 派生的。

下面的示例生成 CS0184:

```
// CS0184.cs
// compile with: /W:1
class MyClass
{
    public static void Main()
    {
        int i = 0;
        if (i is string)    // CS0184
            i++;
    }
}
```

编译器警告(等级 1)CS0420

错误消息

"identifier": 对 volatile 字段的引用不被视为 volatile

volatile 字段不能像正常情况那样使用 **ref** 或 **out** 参数进行传递，因为它在函数的范围内不会被视为 volatile。这也有例外情况，例如当调用互锁 API 时。对于任何警告，您可以使用 [#pragma warning](#) 在您特意使用 volatile 字段作为引用参数的情况（这种情况很少出现）中禁用此警告。

下面的示例生成 CS0420：

```
// CS0420.cs
// compile with: /W:1
using System;

class TestClass
{
    private volatile int i;

    public void TestVolatile(ref int ii)
    {
    }

    public static void Main()
    {
        TestClass x = new TestClass();
        x.TestVolatile(ref x.i);    // CS0420
    }
}
```

编译器警告(等级 1)CS0465

错误消息

引入“Finalize”方法会妨碍析构函数调用。您是要声明析构函数吗？

当您使用签名为 `public virtual void Finalize` 的方法创建类时，将发生此警告。

如果将这种类用作基类并且派生类定义一个析构函数，则该析构函数将重写基类的 `Finalize` 方法，而不是 `Finalize`。

示例

下面的示例生成 CS0465。

```
// CS0465.cs
// compile with: /target:library
class A
{
    public virtual void Finalize() {}    // CS0465
}

// OK
class B
{
    ~B() {}
}
```

编译器警告(等级 1)CS0602

错误消息

"old_feature"功能已被否决。请改用"new_feature"

仍然支持在代码中使用的语言功能 (*old_feature*), 但这种支持可能会在未来的版本中被移除。您应使用推荐的语法 (*new_feature*)。

编译器警告(等级 1)CS0612

错误消息

"member"已过时

类设计器使用 [Obsolete](#) 属性标记了成员。这意味着在类的未来版本中可能不支持该成员。

下面的示例显示访问过时的成员如何生成 CS0612:

```
// CS0612.cs
// compile with: /W:1
using System;

class MyClass
{
    [Obsolete]
    public static void ObsoleteMethod()
    {
    }

    [Obsolete]
    public static int ObsoleteField;
}

class MainClass
{
    static public void Main()
    {
        MyClass.ObsoleteMethod();      // CS0612 here: method is deprecated
        MyClass.ObsoleteField = 0;     // CS0612 here: field is deprecated
    }
}
```

编译器警告(等级 1)CS0626

错误消息

方法、运算符或访问器“method”被标记为 external 并且没有属性。请考虑添加 DllImport 属性以指定外部实现。

标记为 **extern** 的方法也应使用属性(如 [DllImport](#) 属性)进行标记。

属性指定在哪个位置实现方法。在运行时，程序将需要此信息。

下面的示例生成 CS0626：

```
// CS0626.cs
// compile with: /warnaserror
using System.Runtime.InteropServices;

public class MyClass
{
    static extern public void mf(); // CS0626
    // try the following line
    // [DllImport("mydll.dll")] static extern public void mf();

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS0658

错误消息

"attribute modifier"不是可识别的属性位置。该块中的所有属性都将被忽略。

指定了无效的属性修饰符。有关更多信息, 请参见[属性目标](#)。

下面的示例生成 CS0658:

```
// CS0658.cs
using System;
public class TestAttribute : Attribute{}
[badAttributeLocation: Test]    // CS0658, badAttributeLocation is invalid
class ClassTest
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS0672

错误消息

成员“member1”重写已过时的成员“member2”。向“member1”中添加 Obsolete 属性

编译器找到了标记为 **obsolete** 的方法的 **override**。然而，重写方法本身未标记为 obsolete。如果被调用，重写方法仍将生成 CS0612。

查看您的方法声明并显式指示是否应将方法(及其所有重写)标记为 **obsolete**。

下面的示例生成 CS0672：

```
// CS0672.cs
// compile with: /W:1
class MyClass
{
    [System.Obsolete]
    public virtual void ObsoleteMethod()
    {
    }
}

class MyClass2 : MyClass
{
    public override void ObsoleteMethod() // CS0672
    {
    }
}

class MainClass
{
    static public void Main()
    {
    }
}
```

编译器警告(等级 1)CS0684

错误消息

"interface"接口被标记为"CoClassAttribute"而不是"ComImportAttribute"

如果在接口上指定 **CoClassAttribute**, 则还必须指定 **ComImportAttribute**。

下面的示例生成 CS0684:

```
// CS0684.cs
// compile with: /W:1
using System;
using System.Runtime.InteropServices;

[CoClass(typeof(C))] // CS0684
// try the following line instead
// [CoClass(typeof(C)), ComImport]
interface I
{
}

class C
{
    static void Main() {}
}
```

编译器警告(等级 1)CS0688

错误消息

"method1"具有链接要求, 但重写或实现的"method2"没有链接要求。可能存在安全漏洞。

对派生类方法设置的链接要求, 可以通过调用基类方法轻松规避此要求。若要关闭此安全漏洞, 基类方法同样需要使用该链接要求。有关更多信息, 请参见 [Demand vs. LinkDemand](#)。

示例

下面的示例生成 CS0688。若要解决此警告但不修改基类, 请从重写方法中移除安全属性。这不能解决安全问题。

```
// CS0688.cs
// compile with: /W:1
using System;
using System.Security.Permissions;

class Base
{
    //Uncomment the following line to close the security hole
    //#[FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")]
    public virtual void DoScaryFileStuff()
    {
    }
}

class Derived: Base
{
    #[FileIOPermission(SecurityAction.LinkDemand, All=@"C:\\")] // CS0688
    public override void DoScaryFileStuff()
    {
    }
    static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1030

错误消息

```
#warning:"text"
```

显示用 `#warning` 指令定义的警告文本。

下面的示例显示如何创建用户定义的警告:

```
// CS1030.cs
class Sample
{
    static void Main()
    {
        #warning Let's give a warning here // CS1030
    }
}
```

编译器警告(等级 1)CS1200

错误消息

"无效功能"这一功能已不推荐使用。请改用"有效功能"。

您所尝试使用的功能已不推荐使用。请更新代码以改用有效功能。

编译器警告(等级 1)CS1201

错误消息

"无效功能"这一功能已不推荐使用。请改用"有效功能"。

您所尝试使用的功能已不推荐使用。请更新代码以改用有效功能。

编译器警告(等级 1)CS1202

错误消息

"无效功能"这一功能已不推荐使用。请改用"有效功能"。

您所尝试使用的功能已不推荐使用。请更新代码以改用有效功能。

编译器警告(等级 1)CS1203

“无效功能”这一功能已不推荐使用。请改用“有效功能”。

您所尝试使用的功能已不推荐使用。请更新代码以改用有效功能。

编译器警告(等级 1)CS1522

错误消息

空的 switch 块

编译器检测到了没有 **case** 或 **default** 语句的 **switch** 块。**switch** 块必须有一个或多个 **case** 或 **default** 语句。

下面的示例生成 CS1522:

```
// CS1522.cs
// compile with: /W:1
using System;
class x
{
    public static void Main()
    {
        int i = 6;

        switch(i)    // CS1522
        {
            // add something to the switch block, for example:
            /*
            case (5):
                Console.WriteLine("5");
                return;
            default:
                Console.WriteLine("not 5");
                return;
            */
        }
    }
}
```

编译器警告(等级 1)CS1570

错误消息

"construct"上的 XML 注释出现 XML 格式错误 — "reason"

当使用 [/doc](#) 时, 源代码中的任何注释都必须为 XML 格式。XML 标记若有任何错误, 将导致生成 CS1570。例如:

- 如果向 **cref** 传递字符串(如在 `<exception>` 标记中), 字符串必须用双引号引起。
- 如果使用标记(如没有结束标记的 `<seealso>`), 必须在右尖括号的前面指定正斜杠 (/)。
- 如果需要在说明文本中使用大于号或小于号, 需要用 `>` 或 `<` 表示它们。
- `<include>` 标记上缺少文件或路径属性, 或者它们的格式不正确。

下面的示例生成 CS1570:

```
// CS1570.cs
// compile with: /W:1
namespace ns
{
    // the following line generates CS1570
    /// <summary> returns true if < 5 </summary>
    // try this instead
    // /// <summary> returns true if &lt;5 </summary>

    public class MyClass
    {
        public static void Main ()
        {
        }
    }
}
```

编译器警告(等级 1)CS1574

错误消息

"construct"上的 XML 注释中有无法解析的 cref 属性"item"

传递给 cref 标记的字符串(如在 <exception> 标记中)引用的成员在当前生成环境中不可用。传递给 cref 标记的字符串在语法上必须是成员或字段的正确名称。

有关更多信息, 请参见[建议的文档注释标记](#)。

下面的示例生成 CS1574:

```
// CS1574.cs
// compile with: /W:1 /doc:x.xml
using System;

/// <exception cref="System.Console.WriteLine">An exception class.</exception>    // CS1574
// instead, uncomment and try the following line
// /// <exception cref="System.Console.WriteLine">An exception class.</exception>
class EClass : Exception
{
}

class TestClass
{
    public static void Main()
    {
        try
        {
        }
        catch(EClass)
        {
        }
    }
}
```

编译器警告(等级 1)CS1580

错误消息

XML 注释 cref 属性中的参数“parameter number”的类型无效

当试图引用重载格式的方法时，编译器检测到语法错误。这通常表明所指定的是参数名而不是类型。格式不正确的行将出现在生成的 XML 文件中。

下面的示例生成 CS1580:

```
// CS1580.cs
// compile with: /W:1 /doc:x.xml
using System;

/// <seealso cref="Test(i)"/> // CS1580
// try the following line instead
// /// <seealso cref="Test(int)"/>
public class MyClass
{
    /// <summary>help text</summary>
    public static void Main()
    {

        /// <summary>help text</summary>
        public void Test(int i)
        {

            /// <summary>help text</summary>
            public void Test(char i)
            {
            }
        }
    }
}
```

编译器警告(等级 1)CS1581

错误消息

XML 注释的 cref 属性中的返回类型无效

当试图引用方法时，编译器检测到由于返回类型无效而产生的错误。

示例

下面的示例生成 CS1581:

```
// CS1581.cs
// compile with: /W:1 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <summary>help text</summary>
    public static void Main()
    {
    }

    /// <summary>help text</summary>
    public static explicit operator int(MyClass f)
    {
        return 0;
    }
}

/// <seealso cref="MyClass.explicit operator intt(MyClass)" /> // CS1581
// try the following line instead
// /// <seealso cref="MyClass.explicit operator int(MyClass)" />
public class MyClass2
{}
```

编译器警告(等级 1)CS1584

错误消息

"member"上的 XML 注释中有语法错误的 cref 属性"invalid_syntax"

传递给文档注释标记的参数之一有无效语法。有关更多信息, 请参见[建议的文档注释标记\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1584。

```
// CS1584.cs
// compile with: /W:1 /doc:CS1584.xml
/// <remarks>Test class</remarks>
public class Test
{
    /// <remarks>Called in <see cref="Test.Mai()"/>.</remarks> // CS1584
    // try the following line instead
    // /// <remarks>Called in <see cref="Test.Main()"/>.</remarks>
    public static void Test2() {}

    /// <remarks>Main method</remarks>
    public static void Main() {}
}
```

编译器警告(等级 1)CS1589

错误消息

无法包括文件“file”的 XML 片段“fragment”-- reason

由于指定的 **reason**, 引用文件 (file) 的 `<include>` 标记的语法 (fragment) 不正确。

格式不正确的行将放置在生成的 XML 文件中。

下面的示例生成 CS1589:

```
// CS1589.cs
// compile with: /W:1 /doc:CS1589_out.xml

/// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/' />    // CS1589
// try the following line instead
// /// <include file='CS1589.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1590

错误消息

无效的 XML 包含元素 — 缺少文件属性

缺少传递给 [`<include>`](#) 标记的路径或文档属性, 或者它们不完整。

下面的示例生成 CS1590:

```
// CS1590.cs
// compile with: /W:1 /doc:x.xml

/// <include path='MyDocs/MyMembers[@name="test"]/*' />    // CS1590
// try the following line instead
// /// <include file='x.doc' path='MyDocs/MyMembers[@name="test"]/*' />
class Test
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1592

错误消息

所包含的注释文件中有格式错误的 XML —“reason”

在由 `<include>` 标记指定的文件中发现了报告为 **reason** 的问题。

编译器警告(等级 1)CS1598

错误消息

未能加载 XML 分析器, 原因如下 : "reason"。将不生成 XML 文档文件"file"。

指定了 [/doc](#) 选项, 但编译器未能找到并加载 msxml3.dll。请确保 msxml3.dll 文件已安装并注册。

编译器警告(等级 1)CS1607

错误消息

程序集生成 — reason

在编译的程序集创建阶段生成了警告。

如果要在 32 位平台上生成一个 64 位应用程序，必须确保在目标平台上安装了所有引用的程序集的 64 位版本。

所有特定于 x86 的公共语言运行库 (CLR) 程序集都具有 64 位的对应项(每个 CLR 程序集都存在于所有平台上)。因此，对于 CLR 程序集可以放心地忽略 CS1607。

有关更多信息，请参见 [Al.exe 工具错误和警告](#)。

编译器警告(等级 1)CS1616

错误消息

选项“option”重写源文件或附加模块中给出的属性“attribute”

如果源文件中的程序集属性 `AssemblyKeyFile` 或 `AssemblyKeyName` 与项目属性中指定的 `/keyfile` 或 `/keycontainer` 命令行选项或密钥文件名或密钥容器冲突，则会出现此警告。

在下面的示例中，假定您有一个名为 `cs1616.snk` 的密钥文件。该文件可以用命令行生成：

```
sn -k CS1616.snk
```

下面的示例生成 CS1616：

```
// CS1616.cs
// compile with: /keyfile:cs1616.snk
using System.Reflection;

// To fix the error, remove the next line
[assembly: AssemblyKeyFile("cs1616b.snk")] // CS1616

class C
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1633

错误消息

无法识别的 #pragma 指令

所使用的杂注不是 C# 编译器支持的已知杂注。若要解决此错误，请仅使用受支持的杂注。

下面的示例生成 CS1633:

```
// CS1633.cs
// compile with: /W:1
#pragma unknown // CS1633

class C
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1634

错误消息

应为 disable 或 restore

如果 #pragma warning 子句的格式错误, 例如省略了 disable 或 restore, 则会出现此错误。有关更多信息, 请参见[#pragma warning 主题](#)。

示例

下面的示例生成 CS1634:

```
// CS1634.cs
// compile with: /W:1

#pragma warning // CS1634
// Try this instead:
// #pragma warning disable 0219

class MyClass
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1635

错误消息

警告“warning code”已被全局禁用，无法还原

如果您使用 **/nowarn** 命令行选项或项目设置禁用整个编译单元的警告，但您又使用 `#pragma warning restore` 试图还原该警告，则会发生此警告。若要解决此错误，请移除 **/nowarn** 命令行选项或项目设置，或者移除您通过命令行或项目设置禁用的任何警告的 `#pragma warning restore`。有关更多信息，请参见 [#pragma warning 主题](#)。

下面的示例生成 CS1635：

```
// CS1635.cs
// compile with: /w:1 /nowarn:162

enum MyEnum {one=1,two=2,three=3};

class MyClass
{
    public static void Main()
    {
#pragma warning disable 162

        if (MyEnum.three == MyEnum.two)
            System.Console.WriteLine("Duplicate");

#pragma warning restore 162
    }
}
```

编译器警告(等级 1)CS1645

错误消息

功能“feature”不是标准化 ISO C# 语言规范的一部分，有些编译器可能不接受它

您正在使用的功能不是 ISO 标准的一部分。使用此功能的代码可能无法在其他编译器上进行编译。

```
// CS1645.cs
// compile with: /W:1 /t:module /langversion:ISO-1
[assembly:System.CLSCompliant(false)]
// To supress the warning use the switch: /nowarn:1645
[module:System.CLSCompliant(false)] // CS1645
class Test
{
}
```

编译器警告(等级 1)CS1658

错误消息

"warning text"。另请参见错误"error code"

编译器在用警告重写错误时会发出此警告。有关此问题的信息，请参考上面所提到的错误。若要从 Visual Studio IDE 内查找相应的错误，请使用索引。例如，如果上面的文本是“另请参见错误'CS1037'”，则请在索引中查找 CS1037。

示例

下面的示例生成 CS1658。

```
// CS1658.cs
// compile with: /doc:x.xml
// CS1584 expected
/// <summary>
/// </summary>
public class C
{
    /// <see cref="C.F(params object[])"/> // CS1658
    public static void M()
    {
    }

    /// <summary>
    /// </summary>
    public void F(params object[] o)
    {
    }

    static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1682

错误消息

对类型“type”的引用声明它嵌套在“nested type”中，但是未能找到它

当您导入的引用与其他引用或者您编写的代码不一致时，会发生此错误。发生此错误通常是因为编写了在元数据中引用类的代码，而此后或是删除了该类或是修改了类的定义。

示例

```
// CS1682.cs
// compile with: /target:library /keyfile:mykey.snk
public class A {
    public class N1 {}
}
```

```
// CS1682_b.cs
// compile with: /target:library /reference:CS1682.dll
using System;
public class Ref {

    public static A A1() {
        return new A();
    }

    public static A.N1 N1() {
        return new A.N1();
    }
}
```

```
// CS1682_c.cs
// compile with: /target:library /keyfile:mykey.snk /out:CS1682.dll
public class A {
    public void M1() {}
}
```

下面的示例生成 CS1682。

```
// CS1682_d.cs
// compile with: /reference:CS1682.dll /reference:CS1682_b.dll /W:1
// CS1682 expected
class Tester {
    static void Main()
    {
        Ref.A1().M1();
    }
}
```

编译器警告(等级 1)CS1683

错误消息

对类型“Type Name”的引用声称在此程序集中定义了该类型，但源代码或任何添加的模块中并未定义该类型

当您导入的程序集反过来包含对您当前正在编译的程序集的引用，而正在编译的程序集不包含任何与该引用匹配的内容时，会出现此错误。导致此情况的一种操作是，编译您的程序集，它最初确实包含所导入的程序集引用的成员。然后更新您的程序集，不小心移除了导入的程序集所引用的成员。

编译器警告(等级 1)CS1684

错误消息

对类型“Type Name”的引用声明它是在“Namespace”中定义的，但却没有找到它

当一个命名空间中的引用引用了一个类型，并声明该类型存在于另一个命名空间中，但该类型并不存在时，可能导致此错误。例如，mydll.dll 声明类型 A 存在于 yourdll.dll 中，但 yourdll.dll 中并没有这种类型。导致此错误的一种可能是，您使用的 yourdll.dll 版本太旧，尚未定义 A。

下面的示例生成 CS1684。

示例

```
// CS1684_a.cs
// compile with: /target:library /keyfile:CS1684.key
public class A {
    public void Test() {}
}

public class C2 {}
```

```
// CS1684_b.cs
// compile with: /target:library /r:cs1684_a.dll
// post-build command: del /f CS1684_a.dll
using System;
public class Ref
{
    public static A GetA() { return new A(); }
    public static C2 GetC() { return new C2(); }
}
```

现在，我们重新生成第一个程序集，并在重新编译中对类 C2 的定义不做任何定义。

```
// CS1684_c.cs
// compile with: /target:library /keyfile:CS1684.key /out:CS1684_a.dll
public class A {
    public void Test() {}
}
```

此模块通过标识符 Ref 引用第二个模块。但是第二个模块包含对类 C2 的引用，由于上一步中的编译，该类已不存在，所以此模块的编译返回 CS1684 错误信息。

```
// CS1684_d.cs
// compile with: /reference:cs1684_a.dll /reference:cs1684_b.dll
// CS1684 expected
class Tester
{
    public static void Main()
    {
        Ref.GetA().Test();
    }
}
```

编译器警告(等级 1)CS1685

错误消息

预定义类型“System.type name”是在全局别名的多个程序集中定义的;请使用“文件名”中的定义

当在两个程序集中都找到一个预定义系统类型(例如 System.int32)时,会出现此错误。如果您从两个不同的位置引用 mscorelib(例如尝试并列运行 1.0 和 1.1 版的 .NET Framework),会发生此错误。

编译器仅使用其中一个程序集中的定义。编译器只搜索全局别名,不搜索定义 **/reference** 的库。如果您指定了 **/nostdlib**, 编译器将搜索 [Object](#), 并且以后所有预定义类型的搜索都将在包含 **Object** 的文件中进行。

编译器警告(等级 1)CS1687

错误消息

源文件已超过在 PDB 中可表示的 16,707,565 行的限制, 调试信息将不正确

PDB 和调试器有一些关于文件大小的限制。如果源文件太大, 调试器在超过该限制后将不能正常工作。用户应该要么不发出源文件的调试信息(一种可能的方式是使用 `#line hidden`), 要么找到缩小源文件的办法(一种可能的方式是将源文件分割成多个文件)。他们可能需要使用 **partial** 关键字来分割大类。

编译器警告(等级 1)CS1691

错误消息

"number"不是有效的警告编号

传递给 [#pragma warning](#) 预处理器指令的数字不是有效的警告编号。请验证该数字表示警告，而不是错误或其他字符序列。

示例

下面的示例生成 CS1691。

```
// CS1691.cs
public class C
{
    int i = 1;
    public static void Main()
    {
        C myC = new C();
#pragma warning disable 151 // CS1691
// Try the following line instead:
// #pragma warning disable 1645
        myC.i++;
#pragma warning restore 151 // CS1691
// Try the following line instead:
// #pragma warning restore 1645
    }
}
```

编译器警告(等级 1)CS1692

错误消息

无效数字

许多预处理器指令(例如 `#pragma` 和 `#line`)使用数字作为参数。由于太大、格式错误或者包含非法字符等原因, 这些数字中的一个无效。若要更正此错误, 请更正数字。

示例

下面的示例生成 CS1692。

```
// CS1692.cs

#pragma warning disable a // CS1692
// Try this instead:
// #pragma warning disable 1691

class A
{
    static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1694

错误消息

为预处理器指令指定的文件名无效。文件名太长或者不是有效的文件名。

使用 `#pragma checksum` 预处理器指令时出现此警告。指定的文件名长度超过 256 个字符。若要解决此警告，请使用较短的文件名。

示例

下面的示例生成 CS1694。

编译器警告(等级 1)CS1695

错误消息

无效的 #pragma 校验和语法; 应为 #pragma 校验和"filename" "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}""XXXX..."

您应该很少会遇到此错误, 因为如果您是通过 Code Dom API 生成代码, 校验和通常在运行时插入。

但是, 如果您要在此 **#pragma** 语句中键入内容, 并且错误地键入了 GUID 或校验和, 您就会遇到此错误。编译器的语法检查并不验证您是否输入了正确的 GUID, 但它会检查数字和分隔符的位数是否正确, 以及这些数字是否为十六进制。同样, 它还验证校验和是否包含偶数位数, 以及这些数字是否为十六进制。

示例

下面的示例生成 CS1695。

```
// CS1695.cs

#pragma checksum "12345" // CS1695

public class Test
{
    static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1696

错误消息

应输入单行注释或行尾

编译器要求预处理器指令后面必须跟行尾结束符或单行注释。编译器已完成对有效的预处理器指令的处理，并且遇到了违反此语法规约的内容。

示例

下面的示例生成 CS1696。

```
// CS1696.cs
class Test
{
    public static void Main()
    {
        #pragma warning disable 1030;219    // CS1696
        #pragma warning disable 1030    // OK
    }
}
```

编译器警告(等级 1)CS1697

错误消息

为“file name”提供了不同的校验和值

您为给定的文件指定了一个以上的校验和。当一个项目中有一个以上的同名文件时，调试器使用校验和值确定调试哪个文件。大多数用户不会遇到此错误，但是如果您正在编写生成代码的应用程序，您可能会偶然遇到此错误。若要解决此错误，请确保对任何给定的代码文件仅生成一次校验和。

编译器警告(等级 1)CS1699

错误消息

使用命令行选项“compiler_option”或适当项目设置代替“attribute_name”

为给程序集签名，必须指定密钥文件。在 Microsoft Visual C# 2005 之前，您在源代码中使用 CLR 属性指定了密钥文件。现在，这些属性都被否决了。

在 Microsoft Visual C# 2005 中开始之前，您应使用“项目设计器”的“签名页”或程序集链接器指定密钥文件。

“项目设计器”的“签名页”是首选方法；有关更多信息，请参见[“项目设计器”->“签名”页和管理程序集签名和清单签名](#)。

如何: 使用强名称为程序集签名 使用如下编译器选项：

- [/keyfile\(指定强名称密钥文件\)\(C# 编译器选项\)](#) 代替 [AssemblyKeyFileAttribute](#) 属性。
- [/keycontainer\(指定强名称密钥容器\)\(C# 编译器选项\)](#) 代替 [AssemblyKeyNameAttribute](#)。
- [/delaysign\(延迟为程序集签名\)\(C# 编译器选项\)](#) 代替 [AssemblyDelaySignAttribute](#)。

否决这些属性的原因是：

- 存在安全问题，因为这些属性嵌入在由编译器生成的二进制文件中。因此，每个拥有您的二进制文件的人都会获得存储在其中的密钥。
- 存在可用性问题，因为属性中指定的路径是相对于当前工作目录（此目录在集成开发环境（IDE）中可能会发生更改）或输出目录的。因此，大多数情况下密钥文件有可能是 ...\\mykey.snk。这些属性还增加了项目系统为附属程序集正确签名的难度。当您使用编译器选项代替这些属性时，可以使用密钥的完全限定路径和文件名，而无需在输出文件中嵌入任何内容；项目系统和源代码管理系统可在项目移动时正确处理该完整路径；项目系统可保留密钥文件的项目相对路径，同时仍将完整路径传递给编译器；其他生成程序可通过直接将适当的路径传递给编译器（而不是用正确的属性生成源文件）更加容易地为输出签名。
- 将属性用于友元程序集会影响编译器的效率。当您使用属性时，编译器在必须决定是否授予友元关系时不知道密钥是什么，因此它必须进行推测。在编译的最后，一旦编译器最后知道了密钥，它就能够对推测进行验证。当使用编译器选项指定密钥文件时，编译器可立即决定是否授予友元关系。

示例

下面的示例生成 CS1699。若要解决此错误，请移除该属性并使用 [/delaysign](#) 进行编译。

```
// CS1699.cs
// compile with: /target:library
[assembly:System.Reflection.AssemblyDelaySign(true)] // CS1699
```

请参见

任务

[如何: 使用强名称为程序集签名](#)

参考

[“项目设计器”->“签名”页](#)

其他资源

[管理程序集签名和清单签名](#)

编译器警告(等级 1)CS1707

错误消息

根据新的语言规则, 委托“DelegateName”已绑定到“MethodName1”而不是“MethodName2”

C# 2.0 实现了将委托绑定到方法的新规则。新规则考虑了其他一些以前未考虑的内容。此警告指示委托现在绑定到的方法重载不是它以前绑定到的方法重载。您可能需要验证委托是否确实应该绑定到“MethodName1”而不是“MethodName2”。

有关编译器如何确定将委托绑定到哪个方法的说明, 请参见[委托中的协变和逆变\(C# 编程指南\)](#)。

编译器警告(等级 1)CS1709

错误消息

为预处理器指令指定的文件名为空

您指定了一个包含文件名的预处理器指令，但该文件名是空的。若要解决此警告，请将所需的内容放入该文件。

示例

下面的示例生成 CS1709。

```
// CS1709.cs
class Test
{
    static void Main()
    {
        #pragma checksum "" "{406EA660-64CF-4C82-B6F0-42D48172A799}" "" // CS1709
    }
}
```

编译器警告(等级 1)CS1720

错误消息

由于“generic type”的默认值为 null，因此表达式总会导致 System.NullReferenceException

如果您编写的表达式涉及泛型类型变量的默认值，而该泛型类型变量是引用类型(例如，类)，则将发生此错误。请看下面的表达式：

```
default(T).ToString()
```

由于 `T` 是引用类型，它的默认值为 null，因此试图对其应用 `ToString` 方法将引发 `NullReferenceException`。

示例

类型 `T` 上的类引用约束确保 `T` 是引用类型。

下面的示例生成 CS1720。

```
// CS1720.cs
using System;
public class Tester
{
    public static void GenericClass<T>(T t1) where T : class
    {
        Console.WriteLine(default(T).ToString()); // CS1720
    }
    public static void Main() {}
}
```

编译器警告(等级 1)CS1723

错误消息

"param"上的 XML 注释具有引用类型参数的 cref 属性"attribute"

此错误由引用类型参数的 XML 注释生成。

示例

下面的示例生成 CS1723。

```
// CS1723.cs
// compile with: /t:library /doc:filename.XML
///<summary>A generic list class.</summary>
///<see cref="T" /> // CS1723
// To resolve comment the previous line.
public class List<T>
{
}
```

编译器警告(等级 1)CS1911

错误消息

从匿名方法或迭代器通过“base”关键字访问成员“member”会导致代码无法验证。

使用 **base** 关键字在迭代器的方法体或匿名方法内调用虚函数将导致无法验证的代码。无法验证的代码在部分受信任的环境中将无法运行。

一个针对 CS1911 的解决方法是将虚函数调用移至 Helper 函数。

示例

下面的示例生成 CS1911。

```
// CS1911.cs
// compile with: /W:1
using System;

delegate void D();
delegate D RetD();

class B {
    protected virtual void M() {
        Console.WriteLine("B.M");
    }
}

class Der : B {
    protected override void M() {
        Console.WriteLine("D.M");
    }
}

void Test() { base.M(); }
D Test2() { return new D(base.M); }

public D CallBaseM() {
    return delegate () { base.M(); }; // CS1911

    // try the following line instead
    // return delegate () { Test(); };
}

public RetD DelToBaseM() {
    return delegate () { return new D(base.M); }; // CS1911

    // try the following line instead
    // return delegate () { return Test2(); };
}

class Program {
    public static void Main() {
        Der der = new Der();
        D d = der.CallBaseM();
        d();
        RetD rd = der.DelToBaseM();
        rd();
    }
}
```

编译器警告(等级 1)CS2002

错误消息

源文件“file”被指定多次

源文件名多次传递给了编译器。为生成输出文件，只能将文件指定给编译器一次。

下面的示例生成 CS2002:

```
// CS2002.cs
// compile with: CS2002.cs
public class A
{
    public static void Main(){}
}
```

若要生成该错误，请用以下命令行编译该示例:

```
csc CS2002.cs CS2002.cs
```

编译器警告(等级 1)CS2014

错误消息

编译器选项“old option”已过时, 请使用“new option”

编译器选项的格式被否决。有关更多信息, 请参见 [C# 编译器选项](#)。

编译器警告(等级 1)CS2023

错误消息

因为 /noconfig 是在响应文件中指定的, 所以忽略该选项

在响应文件中指定了 [/noconfig](#) 编译器选项, 这是不允许的。

编译器警告(等级 1)CS3000

错误消息

带有变量参数的方法不符合 CLS

此方法中使用的参数公开了不属于公共语言规范 (CLS) 的功能。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)。

下面的示例生成警告 CS3000。

```
// CS3000.cs
// compile with: /target:library
// CS3000 expected
[assembly:System.CLSCompliant(true)]

public class Test
{
    public void AddABunchOfInts( __arglist ) {}    // CS3000
}
```

编译器警告(等级 1)CS3001

错误消息

参数类型“type”不符合 CLS

public、**protected** 或 **protected internal** 方法必须接受类型符合公共语言规范 (CLS) 的参数。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3001:

```
// CS3001.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public void bad(ushort i)    // CS3001
    {
    }

    private void OK(ushort i)    // OK, private method
    {
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3002

错误消息

"method"的返回类型不符合 CLS

public、**protected** 或 **protected internal** 方法返回的值的类型必须符合公共语言规范 (CLS)。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3002:

```
// CS3002.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort bad()    // CS3002, public method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    private ushort OK()    // OK, private method
    {
        ushort a;
        a = ushort.MaxValue;
        return a;
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3003

错误消息

"variable"的类型不符合 CLS

public、**protected** 或 **protected internal** 变量的类型必须符合公共语言规范 (CLS)。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3003:

```
// CS3003.cs

[assembly:System.CLSCompliant(true)]
public class a
{
    public ushort a1;    // CS3003, public variable
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3004

错误消息

混合的和分解的 Unicode 字符不符合 CLS

为了符合公共语言规范 (CLS), 在 [public](#)、[protected](#) 或 [protectedinternal](#) 标识符中只允许使用组合的 UNICODE 字符。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

编译器警告(等级 1)CS3005

错误消息

只有大小写不同的标识符“identifier”不符合 CLS

只有一个或多个字母的大小写与其他 **public**、**protected** 或 **protected internal** 标识符不同的 **public**、**protected** 或 **protected internal** 标识符不符合公共语言规范 (CLS)。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3003:

```
// CS3005.cs

using System;

[assembly:CLSCompliant(true)]
public class a
{
    public static int a1 = 0;
    public static int A1 = 1;    // CS3005

    public static void Main()
    {
        Console.WriteLine(a1);
        Console.WriteLine(A1);
    }
}
```

编译器警告(等级 1)CS3006

错误消息

只是 `ref`、`out` 或数组秩不同的重载方法“method”不符合 CLS

方法不能基于 `ref` 或 `out` 参数重载，但仍遵守公共语言规范 (CLS)。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3006：若要解决此警告，请注释掉程序集级属性或移除其中一个方法定义。

```
// CS3006.cs

using System;

[assembly: CLSCompliant(true)]
public class MyClass
{
    public void f(int i)
    {
    }

    public void f(ref int i) // CS3006
    {
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3007

错误消息

只是未命名数组类型不同的重载方法“method”不符合 CLS。

如果您有一个接受交错数组的重载方法，并且方法签名的唯一区别是数组的元素类型，则将发生此错误。为避免此错误，可考虑使用矩形数组而不是交错数组，使用一个额外的参数消除函数调用的歧义，重命名一个或多个重载方法，或者如果不需要 CLS 遵从性，则移除 [CLSCompliantAttribute](#) 属性。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3007：

```
// CS3007.cs
[assembly: System.CLSCompliant(true)]
public struct S
{
    public void F(int[][] array) { }
    public void F(byte[][] array) { } // CS3007
    // Try this instead:
    // public void F1(int[][] array) {}
    // public void F2(byte[][] array) {}
    // or
    // public void F(int[,] array) {}
    // public void F(byte[,] array) {}
}
```

编译器警告(等级 1)CS3008

错误消息

标识符“identifier”不符合 CLS

如果 **public**、**protected** 或 **protected internal** 标识符以下划线字符 (_) 开头，它就违反了公共语言规范 (CLS)。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3008：

```
// CS3008.cs

using System;

[assembly:CLSCompliant(true)]
public class a
{
    public static int _a = 0; // CS3008
    // OK, private
    // private static int _a1 = 0;

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3009

错误消息

"type": 基类型" type "不符合 CLS

在标记为符合公共语言规范 (CLS) 的程序集中, 基类型被标记为不必符合 CLS。请移除指定程序集符合 CLS 的属性, 或移除指示类型不符合 CLS 的属性。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3009:

```
// CS3009.cs

using System;

[assembly:CLSCompliant(true)]
[CLSCompliant(false)]
public class B
{
}

public class C : B    // CS3009
{
    public static void Main () {}
}
```

编译器警告(等级 1)CS3010

错误消息

"member": 符合 CLS 的接口必须只有符合 CLS 的成员

在用 `[assembly:CLCSCompliant(true)]` 标记的程序集中，接口包含用 `[CLCSCompliant(false)]` 标记的成员。移除其中一个符合公共语言规范 (CLS) 的属性。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3010:

```
// CS3010.cs

using System;

[assembly:CLCSCompliant(true)]
public interface I
{
    [CLSCompliant(false)]
    int mf();    // CS3010
}

public class C : I
{
    public int mf()
    {
        return 1;
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3011

错误消息

"member": 只有符合 CLS 的成员才能是抽象的

类成员不能既 [abstract](#), 又不符合公共语言规范 (CLS)。CLS 指定应实现所有的类成员。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3011:

```
// CS3011.cs

using System;

[assembly:CLSCompliant(true)]
public abstract class I
{
    [CLSCompliant(false)]
    public abstract int mf();    // CS3011

    // OK
    [CLSCompliant(false)]
    public void mf2()
    {
    }
}

public class C : I
{
    public override int mf()
    {
        return 1;
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3012

错误消息

必须在程序集而不是模块上指定 `CLSCCompliant` 属性，以便启用 CLS 遵从性检查

为使模块可以通过 `[module:System.CLCSCompliant(true)]` 符合公共语言规范 (CLS)，必须使用 [/target:module](#) 编译器选项生成它。有关 CLS 的更多信息，请参见 [公共语言规范](#)。

示例

当不使用 **/target:module** 生成时，下面的示例生成 CS3012：

```
// CS3012.cs
// compile with: /W:1

[module:System.CLCSCompliant(true)] // CS3012
public class C
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3013

错误消息

添加的模块必须用 `CLSCompliant` 属性标记才能与程序集匹配

用 `/target:module` 编译器选项编译的模块添加到了使用 `/addmodule` 的编译中。但是，模块的公共语言规范 (CLS) 遵从性与当前编译的 CLS 状态不一致。

CLS 遵从性由模块属性指示。例如，`[module:CLSCompliant(true)]` 指示模块符合 CLS，而 `[module:CLSCompliant(false)]` 指示模块不符合 CLS。默认值为 `[module:CLSCompliant(false)]`。有关 CLS 的更多信息，请参见 [公共语言规范](#)。

编译器警告(等级 1)CS3014

错误消息

"member"无法标记为符合 CLS, 因为程序集不具有 CLSCompliant 属性

在未指定公共语言规范 (CLS) 遵从性的源代码文件中, 文件中的构造被标记为了符合 CLS。这是不允许的。若要解决此警告, 请将程序集级符合 CLS 的属性添加到文件中(在以下示例中, 取消对包含程序集级属性的代码行的注释)。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3014:

```
// CS3014.cs

using System;

// [assembly:CLSCompliant(true)]
public class I
{
    [CLSCompliant(true)]    // CS3014
    public void mf()
    {
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3015

错误消息

"method signature"没有只使用符合 CLS 类型的可访问的构造函数

为了符合公共语言规范 (CLS), 属性类的参数列表不能包含数组。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 C3015。

```
// CS3015.cs
// compile with: /target:library
using System;

[assembly:CLSCompliant(true)]
public class MyAttribute : Attribute
{
    public MyAttribute(int[] ai) {} // CS3015
    // try the following line instead
    // public MyAttribute(int ai) {}
}
```

编译器警告(等级 1)CS3016

错误消息

作为属性参数的数组不符合 CLS

将数组传递给属性不符合公共语言规范 (CLS)。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3016:

```
// CS3016.cs

using System;

[assembly : CLSCompliant(true)]
[C(new int[] {1, 2})]    // CS3016
// try the following line instead
// [C()]
class C : Attribute
{
    public C()
    {
    }

    public C(int[] a)
    {
    }

    public static void Main ()
    {
    }
}
```

编译器警告(等级 1)CS3017

错误消息

不能在模块上指定与程序集的 CLSCompliant 属性不同的 CLSCompliant 属性

如果您的程序集 CLSCompliant 属性与模块的 CLSCompliant 属性冲突，则会出现此警告。与 CLS 兼容的程序集不能包含与 CLS 不兼容的模块。若要解决此警告，请确保程序集和模块的 CLSCompliant 属性全都为 true 或者全都为 false，或者移除其中一个属性。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3017：

```
// CS3017.cs
// compile with: /target:module

using System;

[module: CLSCompliant(true)]
[assembly: CLSCompliant(false)] // CS3017
// Try this line instead:
// [assembly: CLSCompliant(true)]
class C
{
    static void Main() {}
}
```

编译器警告(等级 1)CS3018

错误消息

"type"是不符合 CLS 的类型"type"的成员，因此不能将其标记为符合 CLS

如果将 CLSCompliant 属性设置为 **true** 的某一嵌套类声明为通过将 CLSCompliant 属性设置为 **false** 来声明的类的成员，将发生此警告。这是不允许的，因为如果嵌套类是不符合 CLS 的某一外部类的成员，它将无法符合 CLS。要解决此警告，请从嵌套类中移除 CLSCompliant 属性，或将其从 **true** 更改为 **false**。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)和[公共语言规范](#)。

示例

下面的示例生成 CS3018。

```
// CS3018.cs
// compile with: /target:library
using System;

[assembly: CLSCompliant(true)]
[CLSCompliant(false)]
public class Outer
{
    [CLSCompliant(true)]    // CS3018
    public class Nested {}

    // OK
    public class Nested2 {}

    [CLSCompliant(false)]
    public class Nested3 {}
}
```

编译器警告(等级 1)CS3022

错误消息

CLSCompliant 属性在应用于参数时无意义。请尝试将该属性应用于方法。

由于 CLS 遵从性规则适用于方法和类型声明，因此不检查方法参数是否符合 CLS 遵从性。

示例

下面的示例生成 CS3022:

```
// CS3022.cs
// compile with: /W:1

using System;

[assembly: CLSCompliant(true)]
[CLSCompliant(true)]
public class C
{
    public void F([CLSCompliant(true)] int i)
    {
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS3023

错误消息

CLSCompliant 属性在应用于返回类型时没有意义。请尝试将该属性应用于方法。

由于 CLS 遵从性规则应用于方法和类型声明，因此不检查函数返回类型是否符合 CLS 遵从性。

示例

下面的示例生成警告 CS3023：

```
// C3023.cs

[assembly:System.CLSCompliant(true)]
public class Test
{
    [return:System.CLSCompliant(true)] // CS3023
    // Try this instead:
    // [method:System.CLSCompliant(true)]
    public static int Main()
    {
        return 0;
    }
}
```

编译器警告(等级 1)CS3026

错误消息

符合 CLS 的字段“field”不能是可变字段

可变变量不应符合 CLS。

示例

下面的示例生成 CS3026。

```
// CS3026.cs
[assembly:System.CLSCompliant(true)]
public class Test
{
    public volatile int v0 =0;    // CS3026
        // To resolve remove the CLS-CCompliant attribute.
    public static void Main() { }
}
```

编译器警告(等级 1)CS5000

错误消息

未知的编译器选项"/option"

指定了无效的[编译器选项](#)。

编译器警告(等级 2)CS0108

错误消息

"member1"会隐藏继承的成员"member2"。如果打算隐藏, 请使用 new 关键字。

声明变量所使用的名称与基类中的变量相同。但没有使用 new 关键字。此警告通知您应使用 new; 声明变量时将按已在声明中使用了 new 来处理。

下面的示例生成 CS0108:

```
// CS0108.cs
// compile with: /W:2
using System;

namespace x
{
    public class clk
    {
        public int i = 1;
    }

    public class cly : clk
    {
        public static int i = 2;    // CS0108, use the new keyword
        // the compiler parses the previous line as if you had specified:
        // public static new int i = 2;

        public static void Main()
        {
            Console.WriteLine(i);
        }
    }
}
```

编译器警告(等级 2)CS0114

错误消息

"function1"将隐藏继承的成员"function2"。若要用当前方法重写该实现, 请添加 override 关键字。否则, 添加关键字 new。

类中的声明与基类中的声明冲突, 以致基类成员将被隐藏。

有关更多信息, 请参见 [base](#)。

下面的示例生成 CS0114:

```
// CS0114.cs
// compile with: /W:2 /warnaserror
abstract public class clx
{
    public abstract void f();
}

public class cly : clx
{
    public void f() // CS0114, hides base class member
    // try the following line instead
    // override public void f()
    {

}

    public static void Main()
    {
}
}
```

编译器警告(等级 2)CS0162

错误消息

检测到无法访问的代码

编译器检测到永远不执行的代码。

下面的示例生成 CS0162:

```
// CS0162.cs
// compile with: /W:2
public class A
{
    public static void Main()
    {
        goto lab1;
        {
            // The following statements cannot be reached:
            int i = 9;    // CS0162
            i++;
        }
        lab1:
        {
        }
    }
}
```

编译器警告(等级 2)CS0164

错误消息

这个标签尚未被引用

声明了标签, 但从未使用过。

下面的示例生成 CS0164:

```
// CS0164.cs
// compile with: /W:2
public class a
{
    public int i = 0;

    public static void Main()
    {
        int i = 0;    // CS0164
        l1: i++;
        // the following lines resolve this error
        // if(i < 10)
        //     goto l1;
    }
}
```

编译器警告(等级 2)CS0251

错误消息

用负索引对数组进行索引(数组索引总是从零开始)

不要在数组中使用负数进行索引。

下面的示例生成 CS0251:

```
// CS0251.cs
// compile with: /W:2
class MyClass
{
    public static void Main()
    {
        int[] myarray = new int[] {1,2,3};
        try
        {
            myarray[-1]++; // CS0251
            // try the following line instead
            // myarray[1]++;
        }
        catch (System.IndexOutOfRangeException e)
        {
            System.Console.WriteLine("{0}", e);
        }
    }
}
```

编译器警告(等级 2)CS0252

错误消息

可能非有意的引用比较;若要获取值比较,请将左边强制转换为类型“type”

编译器正在进行引用比较。如果要比较字符串的值,请将表达式的左边转换为 *type*。

下面的示例生成 CS0252:

```
// CS0252.cs
// compile with: /W:2
using System;

class MyClass
{
    public static void Main()
    {
        string s = "11";
        object o = s + s;

        bool b = o == s;      // CS0252
        // try the following line instead
        // bool b = (string)o == s;
    }
}
```

编译器警告(等级 2)CS0253

错误消息

可能非有意的引用比较;若要获取值比较,请将右边的值强制转换为类型“type”

编译器正在进行引用比较。如果要比较字符串的值,请将表达式的右边转换为 *type*。

下面的示例生成 CS0253:

```
// CS0253.cs
// compile with: /W:2
using System;
class MyClass
{
    public static void Main()
    {
        string s = "11";
        object o = s + s;

        bool c = s == o;    // CS0253
        // try the following line instead
        // bool c = s == (string)o;
    }
}
```

编译器警告(等级 2)CS0278

错误消息

"type"不实现"pattern name"模式。"method name"与"method name"混淆。

在 C# 中有若干语句依赖定义的模式, 如 **foreach** 和 **using**。例如, **foreach** 依赖于实现"enumerable"模式的集合类。

当编译器由于多义性无法进行匹配时, 会发生 CS0278。例如, "enumerable"模式要求有一个名为 **MoveNext** 的方法, 而您的代码可能包含两个名为 **MoveNext** 的方法。编译器将尝试找到一个接口来使用, 但建议您找出导致二义性的原因并加以解决。

有关更多信息, 请参见[如何: 使用 foreach 访问集合类\(C# 编程指南\)](#)。

示例

下面的示例生成 CS0278。

```
// CS0278.cs
using System.Collections.Generic;
public class myTest
{
    public static void TestForeach<W>(W w)
        where W: IEnumerable<int>, IEnumerable<string>
    {
        foreach (int i in w) {}    // CS0278
    }
}
```

编译器警告(等级 2)CS0279

错误消息

"type name"不实现"pattern name"模式。"method name"是静态的或不是公共的。

在 C# 中有几个依赖于已定义模式的语句, 例如 **foreach** 和 **using**。例如, **foreach** 依赖于实现枚举模式的集合类。当由于一个方法被声明为 **static** 或者不是 **public** 而导致编译器无法完成匹配时, 将出现此错误。模式中的方法必须是类的实例, 并且必须是公共的。

示例

下面的示例生成 CS0279:

```
// CS0279.cs

using System;
using System.Collections;

public class myTest : IEnumerable
{
    IEnumerator IEnumerable.GetEnumerator()
    {
        yield return 0;
    }

    internal IEnumerator GetEnumerator()
    {
        yield return 0;
    }

    public static void Main()
    {
        foreach (int i in new myTest()) {} // CS0279
    }
}
```

编译器警告(等级 2)CS0280

错误消息

"type"不实现"pattern name"模式。"method name"有签名错误。

C# 中的两个语句 **foreach** 和 **using** 分别依赖预定义模式"collection"和"resource"。当由于方法的签名不正确, 而使编译器无法将这些语句中的一个与它的模式相匹配时, 将出现此警告。例如, "collection"模式要求有一个名为 [MoveNext](#) 的方法, 该方法没有参数并且返回 **boolean**。您的代码可以包含一个有参数或者可能返回对象的 **MoveNext** 方法。

"resource"模式和 **using** 提供另一个示例。"resource"模式需要 [Dispose](#) 方法; 如果您以同样的名称定义属性, 您将得到此警告。

要解决此警告, 请确保类型中的方法签名与模式中相应方法的签名相匹配, 并确保您没有与模式所需的方法具有相同名称的属性。

示例

下面的示例生成 CS0280。

```
// CS0280.cs
using System;
using System.Collections;

public class ValidBase: IEnumerable
{
    IEnumerator IEnumerable.GetEnumerator()
    {
        yield return 0;
    }

    internal IEnumerator GetEnumerator()
    {
        yield return 0;
    }
}

class Derived : ValidBase
{
    // field, not method
    new public int GetEnumerator();
}

public class Test
{
    public static void Main()
    {
        foreach (int i in new Derived()) {}    // CS0280
    }
}
```

编译器警告(等级 2)CS0435

错误消息

"file_2"中的命名空间"namespace"与"file_1"中的导入类型"type"冲突。使用命名空间。

当源文件(file_2)中的命名空间与file_1中的导入类型冲突时会发出此警告。编译器使用源文件中的命名空间。

下面的示例生成CS0435:

首先编译此文件:

```
// CS0435_1.cs
// compile with: /t:library
public class Util
{
    public class A { }
}
```

然后, 编译此文件:

```
// CS0435_2.cs
// compile with: /r:CS0435_1.dll

using System;

namespace Util
{
    public class A { }
}

public class Test
{
    public static void Main()
    {
        Console.WriteLine(typeof(Util.A)); // CS0435
    }
}
```

编译器警告(等级 2)CS0436

错误消息

"file_2"中的类型" type "与导入的类型" type "冲突。使用"file_2"中的类型

当源文件 (file_2) 中的类型与 file_1 中的导入类型冲突时会发出此警告。编译器使用源文件中的类型。

示例

```
// CS0436_a.cs
// compile with: /target:library
public class A {
    public void Test() {
        System.Console.WriteLine("CS0436_a");
    }
}
```

下面的示例生成 CS0436。

```
// CS0436_b.cs
// compile with: /reference:CS0436_a.dll
// CS0436 expected
public class A {
    public void Test() {
        System.Console.WriteLine("CS0436_b");
    }
}

public class Test
{
    public static void Main()
    {
        A x = new A();
        x.Test();
    }
}
```

编译器警告(等级 2)CS0437

错误消息

"file_2"中的类型"type"与"file_1"中的导入命名空间"namespace"冲突。将使用该类型。

当源文件 file_2 中的类型与 file_1 中的导入命名空间冲突时发出此警告。编译器使用源文件中的类型。

示例

```
// CS0437_a.cs
// compile with: /target:library
namespace Util
{
    public class A {
        public void Test() {
            System.Console.WriteLine("CS0437_a.cs");
        }
    }
}
```

下面的示例生成 CS0437。

```
// CS0437_b.cs
// compile with: /reference:CS0437_a.dll /W:2
// CS0437 expected
class Util
{
    public class A {
        public void Test() {
            System.Console.WriteLine("CS0437_b.cs");
        }
    }
}

public class Test
{
    public static void Main()
    {
        Util.A x = new Util.A();
        x.Test();
    }
}
```

编译器警告(等级 2)CS0440

错误消息

由于“global::”始终引用全局命名空间而非别名，所以不建议定义名为“global”的别名

当您定义名为 global 的别名时会发出此警告。

示例

下面的示例生成 CS0440:

```
// CS0440.cs
// Compile with: /W:1

using global = MyClass;    // CS0440
class MyClass
{
    static void Main()
    {
        // Note how global refers to the global namespace
        // even though it is redefined above.
        global::System.Console.WriteLine();
    }
}
```

编译器警告(等级 2)CS0444

错误消息

在“System namespace 1”中未找到预定义类型“type name 1”，却在“System namespace 2”中找到了
在编译器预期的位置未找到诸如 [Int32](#) 这样的预定义对象，而是在“System namespace 2”中找到了这类对象。
此错误可能表明 .NET Framework 没有正确安装。若要修复此错误，请重新安装 .NET Framework。
如果您正在编写您自己的基类库，可能也会遇到此错误。这种情况下，如果要解决此错误，请重新生成 mscorelib。

编译器警告(等级 2)CS0458

错误消息

表示式的结果始终是“type name”类型的“null”

此警告由始终导致 **null** 的 **nullable** 表达式所致。

下面的代码生成警告 CS0458。

示例

此示例阐释了一些因 **nullable** 类型导致此错误的不同运算。

```
// CS0458.cs
using System;
public class Test
{
    public static void Main()
    {
        int a = 5;
        int? b = a + null;      // CS0458
        int? qa = 15;
        b = qa + null;         // CS0458
        b -= null;              // CS0458
        int? qa2 = null;
        b = qa2 + null;         // CS0458
        qa2 -= null;             // CS0458
    }
}
```

编译器警告(等级 2)CS0464

错误消息

与类型“type”的空比较总是产生“false”

当您在可为空值的变量与空值之间进行比较时, 如果比较不是 `==` 或 `!=`, 将出现此警告。若要解决此错误, 请验证您是否确实要检查值是否为 `null`。类似 `i == null` 这样的比较结果既可以为 `true`, 也可以为 `false`。类似 `i > null` 这样的比较结果始终为 `false`。

示例

下面的示例生成 CS0464。

```
// CS0464.cs
class MyClass
{
    public static void Main()
    {
        int? i = 0;
        if (i < null) ;    // CS0464

        i++;
    }
}
```

编译器警告(等级 2)CS0467

错误消息

方法“method”和非方法“non-method”之间存在多义性。正在使用方法组。

如果继承的成员具有相同的签名，但来自不同的接口，则会导致多义性错误。

示例

下面的示例生成 CS0467。

```
// CS0467.cs
interface IList
{
    int Count { get; set; }
}
interface ICounter
{
    void Count(int i);
}

interface IListCounter : IList, ICounter {}

class Driver
{
    void Test(IListCounter x)
    {
        x.Count = 1;
        x.Count(1);    // CS0467
        // To resolve change the method name "Count" to another name.
    }

    static void Main()
    {
    }
}
```

编译器警告(等级 2)CS0618

错误消息

"member"已过时 :"text"

类成员是用 [Obsolete](#) 属性标记的, 以致在引用该类成员时会发出警告。

下面的示例生成 CS0618:

```
// CS0618.cs
// compile with: /W:2
using System;

public class C
{
    [Obsolete("Use newMethod instead", false)] // warn if referenced
    public static void m2()
    {
    }

    public static void newMethod()
    {
    }
}

class MyClass
{
    public static void Main()
    {
        C.m2(); // CS0618
    }
}
```

编译器警告(等级 2)CS0652

错误消息

与整数常数比较无意义;该常数不在"type"类型的范围之内

编译器检测到在常数和变量之间的比较中, 常数超出了变量的范围。

下面的示例生成 CS0652:

```
// CS0652.cs
// compile with: /W:2
public class Class1
{
    private static byte i = 0;
    public static void Main()
    {
        short j = 256;
        if (i == 256)    // CS0652, 256 is out of range for byte
            i = 0;
    }
}
```

编译器警告(等级 2)CS0728

错误消息

对局部变量“variable”(它是 using 或 lock 语句的参数)的赋值可能不正确。

在有些情况下, **using** 或 **lock** 块将导致临时资源泄漏。下面是一个示例:

```
thisType f = null;
using (f)
{
    f = new thisType();
    ...
}
```

在这种情况下, 在 **using** 块的执行结束后将会释放变量 thisType 的原始值(如 null), 但不会释放该块内创建的 thisType 对象, 尽管它最终会被回收。

要解决此错误, 请使用以下窗体:

```
using (thisType f = new thisType())
{
    ...
}
```

在这种情况下, 将释放新分配的 thisType 对象。

示例

以下代码将生成警告 CS0728。

```
// CS0728.cs

using System;
public class ValidBase : IDisposable
{
    public void Dispose() { }
}

public class Logger
{
    public static void dummy()
    {
        ValidBase vb = null;
        using (vb)
        {
            vb = null; // CS0728
        }
        vb = null;
    }
    public static void Main() { }
}
```

编译器警告(等级 2)CS1571

错误消息

"construct"上的 XML 注释中对"parameter"有重复的 param 标记

当使用 [/doc](#) 编译器选项时, 找到了同一方法参数的多个注释。移除其中的一个重复行。

下面的示例生成 CS1571:

```
// CS1571.cs
// compile with: /W:2 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <param name='Int1'>Used to indicate status.</param>
    /// <param name='Char1'>An initial.</param>
    /// <param name='Int1'>Used to indicate status.</param> // CS1571
    public static void MyMethod(int Int1, char Char1)
    {
    }

    /// <summary>help text</summary>
    public static void Main ()
    {
    }
}
```

编译器警告(等级 2)CS1572

错误消息

"construct"上的 XML 注释有"parameter"的 param 标记, 但是没有该名称的参数

当使用 [/doc](#) 编译器选项时, 为某个参数指定了注释, 但方法没有这样的参数。更改传递给名称属性的值或移除其中的一个注释行。

下面的示例生成 CS1572:

```
// CS1572.cs
// compile with: /W:2 /doc:x.xml

/// <summary>help text</summary>
public class MyClass
{
    /// <param name='Int1'>Used to indicate status.</param>
    /// <param name='Char1'>Used to indicate status.</param>
    /// <param name='Char2'>???</param> // CS1572
    public static void MyMethod(int Int1, char Char1)
    {
    }

    /// <summary>help text</summary>
    public static void Main ()
    {
    }
}
```

编译器警告(等级 2)CS1587

错误消息

XML 注释没有放在有效语言元素上

并非所有语言元素上都能使用建议的文档注释标记。例如，命名空间中不允许使用标记。有关 XML 注释的更多信息，请参见[建议的文档注释标记\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1587:

```
// CS1587.cs
// compile with: /W:2 /doc:x.xml

/// <summary>test</summary> // CS1587, tag not allowed on namespace
namespace MySpace
{
    class MyClass
    {
        public static void Main()
        {
        }
    }
}
```

编译器警告(等级 2)CS1668

错误消息

"path string"中指定的搜索路径"path"无效 --"system error message"

命令行中提供的 [/lib](#) 路径无效, 或者 LIB 环境变量中的路径无效。检查所使用的路径, 确认它是否存在并且可访问。单引号中的错误消息就是从操作系统返回的错误。

此错误的一个已知原因是从某台计算机上卸载 Visual Studio .NET 2002 或 Visual Studio .NET 2003 时, 该计算机上也安装了 Visual Studio 2005。请执行下面的步骤修复此错误:

1. 右击“我的电脑”, 再单击“属性”。
2. 单击“高级”选项卡, 再单击“环境变量”。
3. 单击上窗格中的 LIB 条目, 确保其指向 v2.0 lib 文件。

如果已将 Visual Studio 2005 安装到默认位置, 则其完整路径为 C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Lib。

编译器警告(等级 2)CS1698

错误消息

循环程序集引用“AssemblyName1”与输出程序集名称“AssemblyName2”不匹配。请尝试添加对“AssemblyName1”的引用，或者更改输出程序集的名称以使其匹配。

程序集引用不正确时将发生 CS1698。当引用的程序重新编译时会发生这种情况。若要解决此问题，请不要替换一个其自身就是所引用的程序集的依赖项的程序集。

示例

```
// CS1698_a.cs
// compile with: /target:library /keyfile:mykey.snk
[assembly:System.Reflection.AssemblyVersion("2")]
public class CS1698_a {}
```

```
// CS1698_b.cs
// compile with: /target:library /reference:CS1698_a.dll /keyfile:mykey.snk
public class CS1698_b : CS1698_a {}
```

下面的示例生成 CS1698。

```
// CS1698_c.cs
// compile with: /target:library /out:cs1698_a.dll /reference:cs1698_b.dll /keyfile:mykey.snk
// CS1698 expected
[assembly:System.Reflection.AssemblyVersion("3")]
public class CS1698_c : CS1698_b {}
public class CS1698_a {}
```

编译器警告(等级 2)CS1701

错误消息

假定程序集引用“Assembly Name #1”与“Assembly Name #2”匹配，您可能需要提供运行时策略

这两个程序集在发行号和/或版本号上有差异。为保证一致，必须在应用程序的 .config 文件中指定指令，并提供程序集的正确强名称，如以下代码示例所示。

示例

下面的多文件示例使用两个不同的外部别名引用一个程序集。第一个示例生成用于创建程序集 CS1701_d 的代码的较早版本。

```
// CS1701_a.cs
// compile with: /target:library /out:cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly: AssemblyVersion("1.0")]
public class A {
    public void M1() {}
}

public class C1 {}
```

这是创建较新版本的程序集 CS1701_d 的代码。注意，由于这两个版本的输出文件同名，需要将较新版本编译到不同于较早版本的目录中。

```
// CS1701_b.cs
// compile with: /target:library /out:c:\\cs1701_d.dll /keyfile:mykey.snk
using System.Reflection;
[assembly: AssemblyVersion("2.0")]
public class A {
    public void M2() {}
    public void M1() {}
}

public class C2 {}
public class C1 {}
```

此示例设置外部别名 A1 和 A2。

```
// CS1701_c.cs
// compile with: /target:library /reference:A2=c:\\cs1701_d.dll /reference:A1=cs1701_d.dll

extern alias A1;
extern alias A2;
// using System;
using a1 = A1::A;
using a2 = A2::A;

public class Ref {
    public static a1 A1() { return new a1(); }
    public static a2 A2() { return new a2(); }

    public static A1::C1 M1() { return new A1::C1(); }
    public static A2::C2 M2() { return new A2::C2(); }
}
```

此示例使用 A 的两个不同别名调用方法。下面的示例生成 CS1701。

```
// CS1701_d.cs
// compile with: /reference:c:\\CS1701_d.dll /reference:CS1701_c.dll
// CS1701 expected
class Tester {
```

```
public static void Main() {  
    Ref.A1().M1();  
    Ref.A2().M2();  
}  
}
```

编译器警告(等级 2)CS1710

错误消息

"type"上的 XML 注释中有"parameter"的重复 typeparam 标记

泛型类型的文档中包括此类型参数的重复标记。

示例

下面的代码将导致出现警告 CS1710。

```
// CS1710.cs
// To resolve this warning, delete one of the duplicate <typeparam>'s.
using System;
class Stack<ItemType>
{
}

/// <typeparam name="MyType">can be an int</typeparam>
/// <typeparam name="MyType">can be an int</typeparam>
class MyStackWrapper<MyType>
{
    // Open constructed type Stack<MyType>.
    Stack<MyType> stack;
    public MyStackWrapper(Stack<MyType> s)
    {
        stack = s;
    }
}

class CMain
{
    public static void Main()
    {
        // Closed constructed type Stack<int>.
        Stack<int> stackInt = new Stack<int>();
        MyStackWrapper<int> MyStackWrapperInt =
            new MyStackWrapper<int>(stackInt);
    }
}
```

编译器警告(等级 2)CS1711

错误消息

"type"上的 XML 注释中有"parameter"的 typeparam 标记, 但是不存在具有该名称的类型参数

泛型类型的文档中包含名称错误的类型参数的标记。

示例

下面的代码生成警告 CS1711。

```
// cs1711.cs
// compile with: /doc:cs1711.xml
// CS1711 expected
using System;
///<typeparam name="WrongName">can be an int</typeparam>
class CMain
{
    public static void Main() { }
```

编译器警告(等级 2)CS3019

错误消息

CLS 遵从性检查不会在“type”上执行，因为它在此程序集的外部不可见。

当一个具有 [CLSCompliantAttribute](#) 属性的类型或成员从另一个程序集中不可见时，将出现此警告。若要解决此错误，请移除从其他程序集中不可见的所有类或成员的属性，或者使类型或成员可见。有关 CLS 遵从性的更多信息，请参见[编写符合 CLS 的代码](#)。

示例

下面的示例生成 CS3019：

```
// CS3019.cs
// compile with: /W:2

using System;

[assembly: CLSCompliant(true)]

// To fix the error, remove the next line
[CLSCompliant(true)] // CS3019
class C
{
    [CLSCompliant(false)] // CS3019
    void Foo()
    {
    }

    static void Main()
    {
    }
}
```

请参见

[概念](#)

[公共语言规范](#)

编译器警告(等级 2)CS3021

错误消息

由于程序集没有 CLSCompliant 属性, 因此“type”不需要 CLSCompliant 属性

如果 `[CLSCompliant(false)]` 出现在程序集中的类上, 而该程序集没有将程序集级 CLSCompliant 属性设置为 true(即行 `[assembly: CLSCompliant(true)]`), 则会出现此警告。由于程序集没有声明自身符合 CLS, 因此程序集中的任何对象均不需要声明自身不符合 CLS, 因为已假定不符合 CLS。有关 CLS 遵从性的更多信息, 请参见[编写符合 CLS 的代码](#)。

若要消除此警告, 请移除该属性或添加程序集级属性。

示例

下面的示例生成 CS3021:

```
// CS3021.cs
using System;
// Uncomment the following line to declare the assembly CLS Compliant,
// and avoid the warning without removing the attribute on the class.
//[assembly: CLSCompliant(true)]

// Remove the next line to avoid the warning.
[CLSCompliant(false)]           // CS3021
public class C
{
    public static void Main()
    {
    }
}
```

请参见

[概念](#)

[公共语言规范](#)

编译器警告(等级 3)CS0067

错误消息

从未使用过事件“event”

声明了 `event`, 但它从未在声明它的类中使用。

下面的示例生成 CS0067:

```
// CS0067.cs
// compile with: /W:3
using System;
delegate void MyDelegate();

class MyClass
{
    public event MyDelegate evt;    // CS0067
    // uncomment TestMethod to resolve this CS0067
/*
    private void TestMethod()
    {
        if (evt != null)
            evt();
    }
*/
    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0105

错误消息

"namespace"的 using 指令以前在此命名空间中出现过

namespace 只应声明一次, 但却声明了多次; 请移除所有重复的命名空间声明。

下面的示例生成 CS0105:

```
// CS0105.cs
// compile with: /W:3
using System;
using System; // CS0105

public class a
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0168

错误消息

声明了变量“var”，但从未使用过

当声明了变量但不使用时，编译器会发出警告。

下面的示例生成两个 CS0168 警告：

```
// CS0168.cs
// compile with: /W:3
public class clx
{
    public int i;
}

public class clz
{
    public static void Main()
    {
        int j = 0;      // CS0168, uncomment the following line
        // j++;
        clx a;         // CS0168, try the following line instead
        // clx a = new clx();
    }
}
```

编译器警告(等级 3)CS0169

错误消息

私有字段“class member”从未使用过

声明了私有变量，但从未引用过。在声明了类的私有成员却没有使用它时，通常将生成该警告。

下面的示例生成 CS0169：

```
// compile with: /W:3
using System;
public class ClassX
{
    int i;    // CS0169, i is not used anywhere

    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0219

错误消息

变量“variable”已赋值，但其值从未使用过

在声明了变量并对其赋值但不使用它时，编译器会发出第 3 级警告。

下面的示例生成 CS0219：

```
// CS0219.cs
// compile with: /W:3
public class MyClass
{
    public static void Main()
    {
        int a = 0;    // CS0219
    }
}
```

编译器警告(等级 3)CS0282

错误消息

在分部类或结构“type”的多个声明中的字段之间没有已定义的排序方式。要指定排序方式，所有实例字段必须位于同一声明中。

要解决此错误，请将所有的成员变量放在一个单独的分部类定义中。

遇到此错误的通常情况是：在多个位置定义分部 **struct**，其中某些成员变量在一个定义中，而另一些成员变量在其他定义中。

下面的代码生成 CS0282。

示例

此代码包含对 **struct** 的描述。通过以下命令，在一个步骤内将这两个模块编译到一起：

```
csc /targt:library cs0282_1.cs cs0282_2.cs
```

```
partial struct A
{
    int i;
}
```

此代码包含对同一 **struct** 的冲突性描述。

```
partial struct A
{
    int j;
}
```

编译器警告(等级 3)CS0414

错误消息

私有字段“field”已被赋值，但该值从未使用过

如果您的类型中有一个已被赋值的私有字段，但该值以后从未被读取过，将出现此警告。不必要的赋值可能影响性能。

下面的示例生成 CS0414：

```
// CS0414
// compile with: /W3
class C
{
    private int i = 1; // CS0414

    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0419

错误消息

cref 属性中有不明确的引用：“Method Name1”。假定为“Method Name2”，但可能还与其他重载匹配，包括“Method Name3”。

在代码的 XML 文档注释中，无法解析引用。如果重载了方法，或找到两个具有相同名称的不同标识符，则可能出现此警告。若要解决此警告，请使用限定名消除引用的歧义，或用括号将特定的重载括起来。

下面的示例生成 CS0419。

```
// cs0419.cs
// compile with: /doc:x.xml /W:3
interface I
{
    /// text for F(void)
    void F();
    /// text for F(int)
    void F(int i);
}
/// text for class MyClass
public class MyClass
{
    /// <see cref="I.F"/>
    public static void MyMethod(int i)
    {
    }
/* Try this instead:
   /// <see cref="I.F(int)" />
   public static void MyMethod(int i)
   {
   }
*/
    /// text for Main
    public static void Main ()
    {
    }
}
```

编译器警告(等级 2)CS0469

错误消息

"goto case"值不可隐式转换为类型"type"

使用 **goto case** 时, 必须将 goto case 的值隐式转换为 switch 的类型。

示例

下面的示例生成 CS0469。

```
// CS0469.cs
// compile with: /W:2
class Test
{
    static void Main()
    {
        char c = (char)180;
        switch (c)
        {
            case (char)127:
                break;

            case (char)180:
                goto case 127;    // CS0469
                // try the following line instead
                // goto case (char) 127;
        }
    }
}
```

编译器警告(等级 3)CS0642

错误消息

空语句可能有错误

条件语句后面的分号可能导致代码不按预期执行。

您可以使用 **/nowarn** 编译器选项或 **#pragmas warning** 来禁用此警告; 有关更多信息, 请参见 [/nowarn\(取消显示指定警告\)\(C# 编译器选项\)](#) 或 [#pragma warning\(C# 参考\)](#)。

下面的示例生成 CS0642:

```
// CS0642.cs
// compile with: /W:3
class MyClass
{
    public static void Main()
    {
        int i;

        for (i = 0; i < 10; i += 1);    // CS0642 semicolon intentional?
        {
            System.Console.WriteLine (i);
        }
    }
}
```

编译器警告(等级 3)CS0659

错误消息

"class"重写 Object.Equals(object o) 但不重写 Object.GetHashCode()

编译器检测到 **Equals** 函数的重写, 但没有检测到 **GetHashCode** 的重写。重写 **Equals** 意味着也要重写 **GetHashCode**。

有关更多信息, 请参见

- [Hashtable](#).
- [Equals 和相等运算符 \(==\) 的实现准则](#)
- [实现 Equals 方法](#)
- [GetHashCode](#)

下面的示例生成 CS0659:

```
// CS0659.cs
// compile with: /W:3 /target:library
class Test
{
    public override bool Equals(object o) { return true; } // CS0659
}

// OK
class Test2
{
    public override bool Equals(object o) { return true; }
    public override int GetHashCode() { return 0; }
}
```

编译器警告(等级 3)CS0660

错误消息

"class" 定义运算符 == 或运算符 !=, 但不重写 Object.Equals(object o)

编译器检测到用户定义的相等运算符或不相等运算符, 但没有检测到 **Equals** 函数的重写。用户定义的相等运算符或不相等运算符意味着也要重写 **Equals** 函数。

下面的示例生成 CS0660:

```
// CS0660.cs
// compile with: /W:3 /warnaserror
class Test // CS0660
{
    public static bool operator == (object o, Test t)
    {
        return true;
    }

    // uncomment the Equals function to resolve
    // public override bool Equals(object o)
    // {
    //     return true;
    // }

    public override int GetHashCode()
    {
        return 0;
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0661

错误消息

"class" 定义运算符 == 或运算符 !=, 但不重写 Object.GetHashCode()

编译器检测到用户定义的相等运算符或不相等运算符, 但没有检测到 **GetHashCode** 函数的重写。用户定义的相等运算符或不相等运算符意味着也要重写 **GetHashCode** 函数。

下面的示例生成 CS0661:

```
// CS0661.cs
// compile with: /W:3
class Test // CS0661
{
    public static bool operator == (object o, Test t)
    {
        return true;
    }

    public static bool operator != (object o, Test t)
    {
        return true;
    }

    public override bool Equals(object o)
    {
        return true;
    }

    // uncomment the GetHashCode function to resolve
    // public override int GetHashCode()
    // {
    //     return 0;
    // }

    public static void Main()
    {
    }
}
```

编译器警告(等级 3)CS0665

错误消息

条件表达式中的赋值总是常数;您是要使用 == 而非 = 吗?

条件表达式使用的是 = 运算符而不是 == 运算符。

下面的示例生成 CS0665:

```
// CS0665.cs
// compile with: /W:3
class Test
{
    public static void Main()
    {
        bool i = false;

        if (i = true)    // CS0665
        // try the following line instead
        // if (i == true)
        {
        }

        System.Console.WriteLine(i);
    }
}
```

编译器警告(等级 3)CS0675

错误消息

按位“或”运算符在带符号扩展操作数上使用;请考虑首先强制转换为较小的无符号类型

编译器隐式地拓宽了并带符号扩展了变量,然后在按位“或”运算中使用了结果值。这可能导致意外的行为。

下面的示例生成 CS0675:

```
// CS0675.cs
// compile with: /W:3
using System;

public class sign
{
    public static void Main()
    {
        int hi = 1;
        int lo = 1;
        long value = (((long)hi) << 32) | lo;           // CS0675
        // try the following line instead
        // long value = (((long)hi) << 32) | ((uint)lo); // correct
    }
}
```

编译器警告(等级 3)CS0693

错误消息

类型参数“type parameter”与外部类型“type”中的类型参数同名

内部类类型参数将隐藏同名的外部类类型参数。若要避免这种情况，请对其中一个类型参数使用其他名称。

示例

下面的示例生成 CS0693。

```
// CS0693.cs
// compile with: /W:3 /target:library
class Outer<T>
{
    class Inner<T> {} // CS0693
    // try the following line instead
    // class Inner<U> {}
}
```

编译器警告(等级 3)CS1700

错误消息

程序集引用 Assembly Name 无效, 无法解析

此警告指示未正确指定属性(如 [InternalsVisibleToAttribute](#))。

有关更多信息, 请参见[友元程序集\(C# 编程指南\)](#)。

示例

下面的示例生成 CS1700。

```
// CS1700.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("app2, Retargetable=f")]    // CS1700
[assembly:InternalsVisibleTo("app2")]    // OK
```

编译器警告(等级 3)CS1702

错误消息

假定程序集引用“Assembly Name #1”和“Assembly Name #2”匹配，您可能需要提供运行时策略

两个程序集引用有不同的内部版本号和/或修订号，所以它们不会自动统一。您可能需要在应用程序的 .config 文件中使用指令，以提供强制统一的运行时策略。

编译器警告(等级 3)CS1717

错误消息

对同一变量进行赋值;您是要给其他变量赋值吗?

当您将一个变量分配给它自身时,例如 `a = a`,会出现此警告。

几个常见的错误可以生成此警告:

- 写入 `a = a` 作为 **if** 语句的条件,例如 `if (a = a)`。您的意思可能是 `if (a == a)`(这总为 true),所以您可以将其简要地写为 `if (true)`。
- 键入错误。您的意思可能是 `a = b`。
- 在一个参数和字段有相同名称的构造函数中,不要使用 **this** 关键字;您的意思可能是 `this.a = a`。

示例

下面的示例生成 CS1717。

```
// CS1717.cs
// compile with: /W:3
public class Test
{
    public static void Main()
    {
        int x = 0;
        x = x;    // CS1717
    }
}
```

编译器警告(等级 3)CS1718

错误消息

对同一变量进行比较;您是要比较其他变量吗?

如果您是要比较其他变量, 则您只需更正语句。

但另一种可能是您要测试真假值, 并通过类似 `if (a == a) (true)` 或 `if (a < a) (false)` 这样的语句来完成此操作。最好简单地说成 `if (true)` 或 `if (false)`。这有两个原因:

- 它更简单;更清楚地表达您的意思。
- 它可以避免混淆:C# 2.0 的一个新特性就是可为空值的值类型, 这类似于 T-SQL(SQL Server 使用的编程语言)中的值 `null`。由于 T-SQL 中使用了三重逻辑, 熟悉 T-SQL 的开发人员可能关心可为空值的类型在类似 `if (a == a)` 这样的表达式上的效果。如果您使用 `true` 或 `false`, 您就可以避免这类可能出现的混淆。

示例

下面的代码生成警告 CS1718。

```
// CS1718.cs
using System;
public class Tester
{
    public static void Main()
    {
        int i = 0;
        if (i == i) { // CS1718.cs
            //if (true) {
                i++;
            //}
        }
    }
}
```

编译器警告(等级 4)CS0028

错误消息

"function declaration"的签名错误, 不能作为入口点

Main 的方法声明无效: 声明所使用的签名无效。**Main** 必须声明为静态的并且它必须返回 [int](#) 或 [void](#)。有关更多信息, 请参见 [Main\(\) 和命令行参数 \(C# 编程指南\)](#)。

下面的示例生成 CS0028:

```
// CS0028.cs
// compile with: /W:4 /warnaserror
public class a
{
    public static double Main(int i) // CS0028
    // Try the following line instead:
    // public static void Main()
    {
    }
}
```

编译器警告(等级 4)CS0078

错误消息

"l"后缀很容易与数字"1"混淆 -- 为清楚起见, 请使用"L"

编译器在检测到转换到的 long 所使用的是小写 l 而不是大写 L 时发出警告。

下面的示例生成 CS0078:

```
// CS0078.cs
// compile with: /W:4
class test {
    public static void TestL(long i)
    {
    }

    public static void TestL(int i)
    {
    }

    public static void Main()
    {
        TestL(25l);    // CS0078
        // try the following line instead
        // TestL(25L);
    }
}
```

编译器警告(等级 4)CS0109

错误消息

成员“member”未隐藏继承成员。不需要关键字 new

类声明包括了 [new](#) 关键字，即使声明不重写基类中的现有声明。可以删除 [new](#) 关键字。

下面的示例生成 CS0109：

```
// CS0109.cs
// compile with: /W:4
namespace x
{
    public class a
    {
        public int i;
    }

    public class b : a
    {
        public new int i;
        public new int j; // CS0109
        public static void Main()
        {
        }
    }
}
```

编译器警告(等级 4)CS0402

错误消息

"identifier": 入口点不能是泛型的或属于泛型类型

在泛型类型中找到入口点。若要移除此警告, 请在非泛型类或结构中实现 Main。

```
// CS0402.cs
// compile with: /W:4
class C<T>
{
    public static void Main() // CS0402
    {

    }
}

class CMain
{
    public static void Main() {}
}
```

编译器警告(等级 4)CS0422

错误消息

不再支持 /incremental 选项

Visual C# 2005 中不支持增量编译(**/incr** 或 **/incremental**)。

编译器警告(等级 4)CS0429

错误消息

检测到无法访问的表达式代码

每当无法访问代码中某个表达式的一部分时，便会发生此错误。在下面的示例中，条件 `false && myTest()` 满足此条件，因为 `myTest()` 方法因 `&&` 运算符左边始终为 `false` 这一事实而永远不会被计算。只要 `&&` 运算符将 **false** 语句计算为 `false`，它便会停止计算，并且永远不会计算右边。

示例

下面的代码生成 CS0429。

```
// CS0429.cs
public class cs0429
{
    public static void Main()
    {
        if (false && myTest()) // CS0429
        // Try the following line instead:
        // if (true && myTest())
        {
        }
        else
        {
            int i = 0;
            i++;
        }
    }

    static bool myTest() { return true; }
}
```

编译器警告(等级 4)CS0628

错误消息

"member":在密封类中声明了新的保护成员

sealed 类无法引入 **protected** 成员，因为没有其他类能够从 **sealed** 类继承并使用 **protected** 成员。

下面的示例生成 CS0628:

```
// CS0628.cs
// compile with: /W:4
sealed class C
{
    protected int i;    // CS0628
}

class MyClass
{
    public static void Main()
    {
    }
}
```

编译器警告(等级 4)CS0649

错误消息

从未对字段“field”赋值，字段将一直保持其默认值“value”

编译器检测到未初始化且从未被赋值的私有字段声明或内部字段声明。

下面的示例生成 CS0649：

```
// CS0649.cs
// compile with: /W:4
using System.Collections;

class MyClass
{
    Hashtable table; // CS0649
    // You may have intended to initialize the variable to null
    // Hashtable table = null;

    // Or you may have meant to create an object here
    // Hashtable table = new Hashtable();

    public void Func(object o, string p)
    {
        // Or here
        // table = new Hashtable();
        table[p] = o;
    }

    public static void Main()
    {
    }
}
```

编译器警告(等级 4)CS1573

错误消息

参数“parameter”在“parameter”的 XML 注释中没有匹配的 param 标记(但其他参数有)

当使用 [/doc](#) 编译器选项时, 为方法中的某些而不是全部参数指定了注释。您可能忘记了输入这些参数的注释。

下面的示例生成 CS1573:

```
// CS1573.cs
// compile with: /W:4
public class MyClass
{
    /// <param name='Int1'>Used to indicate status.</param>
    // enter a comment for Char1?
    public static void MyMethod(int Int1, char Char1)
    {
    }

    public static void Main ()
    {
    }
}
```

编译器警告(等级 4)CS1591

错误消息

缺少对公共可见类型或成员“Type_or_Member”的 XML 注释

指定了 [/doc](#) 编译器选项，但有一个或多个构造没有注释。

下面的示例生成 CS1591：

```
// CS1591.cs
// compile with: /W:4 /doc:x.xml

/// text
public class Test
{
    // /// text
    public static void Main()    // CS1591, remove "://" from previous line
    {
    }
}
```

编译器警告(等级 4)CS1610

错误消息

无法删除用于默认 Win32 资源的临时文件“file”-- resource

在使用 [/win32res](#) 编译器选项时以及当 %TEMP% 目录没有 DELETE 权限时, 该警告指示编译器未能删除它创建的临时文件。

确保对 %TEMP% 目录有读/写/删除权限。

如有必要, 可以手动删除这些文件, 这对 C# 或您的任何程序没有损害。

编译器警告(等级 4)CS1712

错误消息

类型参数“type parameter”在“type”的 XML 注释中没有匹配的 typeparam 标记(但其他类型参数有)

泛型类型的文档缺少 **typeparam** 标记。有关更多信息, 请参见 [<typeparam> \(C# 编程指南\)](#)。

示例

下面的代码生成警告 CS1712。若要解决此错误, 请为类型参数 S 添加一个 **typeparam** 标记。

```
// CS1712.cs
// compile with: /doc:cs1712.xml
using System;
class Test
{
    public static void Main() {}
    /// <param name="j"> This is the j parameter.</param>
    /// <typeparam name="T"> This is the T type parameter.</typeparam>
    public void bar<T,S>(int j) {} // CS1712
}
```

编译器错误 CS1725

错误消息

友元程序集引用“reference”无效。不能在 InternalsVisibleTo 声明中指定版本、区域性、公钥标记或处理器体系结构。

不能在友元程序集引用中添加版本区域性。分部类应对友元程序集可见。

示例

下面的示例生成 CS1725。

```
// CS1725.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("partial01,version=1.1.0.0")]    // CS1725
// try the following line instead
// [assembly:InternalsVisibleTo("partial01")]

partial class TestClass
{
    public static string strBar = "my string";
}
```

编译器错误 CS1726

错误消息

友元程序集引用“reference”无效。强名称签名的程序集必须在其 `InternalsVisibleTo` 声明中指定一个公钥。

强名称签名的程序集只能将使用 `InternalsVisibleToAttribute` 创建的友元程序集访问权限授予其他强签名的程序集。

若要解决 CS1726，请对要授予其友元访问权限的程序集进行签名（赋予其强名称），或不授予友元访问权限。

有关更多信息，请参见[友元程序集 \(C# 编程指南\)](#)。

示例

下面的示例生成 CS1726。

```
// CS1726.cs
// compile with: /keyfile:CS1726.key /target:library
using System.Runtime.CompilerServices;
[assembly:InternalsVisibleTo("UnsignedAssembly")]    // CS1726
// try the following line instead
// [assembly:InternalsVisibleTo("SignedAssembly", PublicKey=00240000048000009400000006020000
0024000052534131000400001000100031d7b6f3abc16c7de526fd67ec2926fe68ed2f9901afbc5f1b6b428bf6
cd9086021a0b38b76bc340dc6ab27b65e4a593fa0e60689ac98dd71a12248ca025751d135df7b98c5f9d09172f7
b62dabdd302b2a1ae688731ff3fc7a6ab9e8cf39fb73c60667e1b071ef7da5838dc009ae0119a9cbff2c581fc0f
2d966b77114b2c4")]
class A {}
```

编译器警告(等级 2)CS0472

错误消息

表达式的结果始终是“value1”，原因是类型“value2”的值永远不会等于类型为“value3”的“null”

如果使用具有恒定空值的运算符，编译器会发出警告。

示例

下面的示例生成 CS0472。

```
public class Test
{
    public static int Main()
    {
        int i = 5;
        int counter = 0;

        // Comparison:
        if (i == null) // CS0472
        // To resolve, use a valid value for i.
            counter++;
        return counter;
    }
}
```

编译器错误 CS1727

错误消息

未经授权无法自动发送错误报告。请访问[“以授权发送错误报告”](#)。

此错误文本中列出的网站介绍如何为 Visual Studio 2005 命令行工具启用自动错误报告。

示例

下面的示例生成 CS1727。

```
// CS1727.cs
// compile with: /errorreport:send
// CS1727 expected
class Test {}
```

编译器错误 CS0735

错误消息

指定为 TypeForwardedTo 属性的参数的类型无效

下面的示例生成 CS0735。

```
// CS735.cs
// compile with: /target:library
using System.Runtime.CompilerServices;
[assembly:TypeForwardedTo(typeof(int[]))]    // CS0735
[assembly:TypeForwardedTo(typeof(string))]    // OK
```

编译器错误 CS1057

错误消息

"member": 静态类不能包含受保护成员

在静态类内部声明受保护成员时，会生成此错误。

示例

下面的示例生成 CS1057。

```
// CS1057.cs
using System;

static class Class1
{
    protected static int x;    // CS1057
    public static void Main()
    {
    }
}
```

编译器警告(等级 1)CS1058

错误消息

前一个 catch 子句已经捕获所有异常。引发的所有非 CLS 异常将包装在 System.Runtime.CompilerServices.RuntimeWrappedException 中

如果 catch() 块在 catch (System.Exception e) 块后没有指定的异常类型，则该属性会导致 CS1058。该警告建议 catch() 块将不捕获任何异常。

如果在 AssemblyInfo.cs 文件中将 **RuntimeCompatibilityAttribute** 设置为 False，则 catch (System.Exception e) 块之后的 catch() 块可捕获非 CLS 异常：[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]。如果未将此属性显式设置为 False，则所有引发的非 CLS 异常将包装为异常，并且 catch (System.Exception e) 块将捕获它们。有关更多信息，请参见[如何: 捕捉非 CLS 异常](#)。

示例

下面的示例生成 CS1058。

```
// CS1058.cs
// CS1058 expected
using System.Runtime.CompilerServices;

// the following attribute is set to true by default in the C# compiler
// set to false in your source code to resolve CS1058
[assembly: RuntimeCompatibilityAttribute(WrapNonExceptionThrows = false)]

class TestClass
{
    static void Main()
    {
        try {}

        catch (System.Exception e) {
            System.Console.WriteLine("Caught exception {0}", e);
        }

        catch {} // CS1058. This line will never be reached.
    }
}
```

C# 术语

访问修饰符

用于限制对类型或类型成员的访问的关键字，如 `private`、`protected`、`internal` 或 `public`。有关更多信息，请参见[访问修饰符](#)。

可访问成员

一种可由给定类型访问的成员。一种类型的可访问成员不一定可由另一种类型访问。有关更多信息，请参见[访问修饰符](#)和[友元程序集](#)。

访问器 (accessor)

设置或检索与属性关联的私有数据成员的值的方法。读写属性具有 `get` 和 `set` 访问器。只读属性只有 `get` 访问器。有关更多信息，请参见[属性](#)。

匿名方法

匿名方法是作为参数传递给[委托](#)的代码块。有关更多信息，请参见[匿名方法](#)。

基类

一种由其他“派生”类继承的类。有关更多信息，请参见[继承](#)。

调用堆栈

运行时从程序的开始位置到当前正在执行的语句的一系列方法调用。

类 (class)

一种描述对象的数据类型。类不仅包含数据，还包含作用于数据的方法。有关更多信息，请参见[类](#)。

构造函数 (constructor)

类或结构中用于初始化该类型的对象的一种特殊方法。有关更多信息，请参见[构造函数](#)。

委托 (delegate)

委托是一种引用方法的类型。一旦为委托分配了方法，委托将与该方法具有完全相同的行为。有关更多信息，请参见[委托](#)。

派生类

一种使用继承获取、扩展或修改其他“基”类的行为和数据的类。有关更多信息，请参见[继承](#)。

析构函数

类或结构中用于准备要由系统销毁的实例的一种特殊方法。有关更多信息，请参见[析构函数](#)。

事件 (event)

类或结构中的一个成员，用于发送更改通知。有关更多信息，请参见[事件](#)。

字段 (field)

类或结构中的一个可直接访问的数据成员。

泛型 (generics)

使用泛型，可以定义使用类型参数定义的类和/或方法。当客户端代码实例化类型时，会指定一个特定的类型作为参数。有关更多信息，请参见[泛型](#)。

IDE

集成开发环境。为各种开发工具（包括编译器、调试器、代码编辑器和设计器）提供统一用户界面的应用程序。

不可变类型 (immutable type)

一种类型，其实例数据（字段和属性）在创建后不会更改。大多数值类型都属于不可变类型。

不可访问的成员

无法由给定类型访问的成员。一种类型的不可访问的成员并不一定对另一种类型是不可访问的。有关更多信息，请参见[访问修饰符](#)。

继承

C# 支持继承，因此从其他类（也称为基类）派生的类将继承相同的方法和属性。继承涉及基类和派生类。有关更多信息，请参见[继承](#)。

接口 (interface)

一种只包含公共方法、事件和委托的签名的类型。继承接口的对象必须实现该接口中定义的所有方法和事件。类或结构可以继承任意数目的接口。有关更多信息，请参见[接口](#)。

iterator

迭代器是这样一种方法，它允许包含集合或数组的类的使用者使用 `foreach`, `in`(C# 参考) 循环访问该集合或数组。

member

在类或结构上声明的字段、属性、方法或事件。

方法

一个提供类或结构的行为的命名代码块。

可变类型 (mutable type)

一种类型，其实例数据（字段和属性）在创建后可以更改。大多数引用类型都属于可变类型。

嵌套类型 (nested type)

在另一个类型的声明内声明的类型。

object

类的实例。对象存在于内存中，具有数据和作用于这些数据的方法。有关更多信息，请参见[对象、类和结构](#)。

属性 (property)

通过访问器访问的数据成员。有关更多信息，请参见[属性](#)。

重构 (refactoring)

重用以前输入的代码。例如，Visual C# 速成版代码编辑器可以智能地重新设置代码的格式，将突出显示的代码块转变成方法。有关更多信息，请参见[重构](#)。

引用类型 (reference type)

一种数据类型。声明为引用类型的变量指向存储数据的位置。有关更多信息，请参见[引用类型](#)。

static

声明为静态的类或方法不必首先使用关键字 `new` 进行实例化就可存在。`Main()` 就属于静态方法。有关更多信息，请参见[静态类和静态类成员](#)。

结构 (struct)

一种复合数据类型，通常用于包含具有某种逻辑关系的多个变量。结构还可以包含方法和事件。结构不支持继承，但支持接口。结构是值类型，而类是引用类型。有关更多信息，请参见[结构](#)。

值类型 (value type)

值类型是在堆栈上分配的数据类型，与在堆上分配的引用类型不同。内置类型（包括数值类型以及结构类型和可空类型）都属于值类型。类类型和字符串类型属于引用类型。有关更多信息，请参见[值类型\(C# 参考\)](#)。

请参见

概念

C# 编程指南

其他资源

C# 参考

C# 语言规范

C# 语言规范的 1.2 和 2.0 版是关于 C# 语法的权威资料。它们包含该语言各个方面的详细信息，包括 Visual C# 产品文档未涉及的许多语法点。

1.2 规范讨论了在 Visual C# 2005 推出之前为该语言添加的功能，而 2.0 规范则讨论了针对 Visual C# 2005 添加的功能。

可以从以下位置获得 Microsoft Word 格式的 C# 语言规范：

- “[Visual C# Developer Center](#)”([Visual C# 开发人员中心](#)) 上的 MSDN online。
- 在 Visual Studio 中的位置是：Microsoft Visual Studio 2005 安装目录下的 VC#\Specifications\1033\ 目录。

如果您的计算机没有安装 Microsoft Word，则可以使用免费的 [Word Viewer 2003](#) 来查看、复制和打印该规范的 Word 版本。

C# 语言规范还以书的形式由 [Addison Wesley](#) 出版发行。

[请参见](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[C# 参考](#)

Visual C# 示例

您可以通过浏览本节中的示例摘要来访问示例代码。每个摘要都包含一个链接，用以打开或复制该示例的文件。另外，.NET Framework SDK 包括技术和应用程序示例，以及演示 .NET Framework 功能和 Visual C# 代码的快速入门教程。

“快速入门教程”是了解 .NET Framework 技术向前沿开发人员提供哪些内容的最快捷方式。快速入门教程是一系列的示例和支持文档，旨在使您能够很快地熟悉 Visual Studio 和 .NET Framework 的语法、结构和强大功能。快速入门教程所包括的内容，除了有许多包含 .NET Framework 技术最引人注目的帮助应用程序外，还有有关 ASP.NET 和 Windows 窗体应用程序的示例。

若要访问快速入门教程，请单击“开始”，指向“程序”，指向“Microsoft .NET Framework SDK v2.0”，然后单击“快速入门教程”。出现“Microsoft .NET Framework SDK 快速入门教程”应用程序的网页。若要运行快速入门教程，请按照该网页上的说明进行操作，这将设置示例数据库并完成安装。有关更多信息，请参见[示例与快速入门](#)。

注意

当打开每个示例的 Visual Studio 解决方案 (.sln) 文件时，Visual C# Express 用户将看到以下消息：“此版本的 Visual Studio 不支持解决方案文件夹。解决方案文件夹‘Solution Items’将显示为不可用。”。虽然在 Visual C# Express 中此文件夹不可用，但您仍能生成并运行对应项目。

本节内容

初级示例

“匿名委托”示例	阐释如何使用未命名委托来减少应用程序的复杂性。
“数组”示例	说明如何使用数组。
“集合类”示例	说明如何生成可与 foreach 语句一起使用的非泛型集合类。
“泛型”示例 (C#)	说明如何生成可与 foreach 语句一起使用的泛型集合类。
“命令行参数”示例	阐释简单的命令行处理和数组索引。
“条件方法”示例	阐释条件方法，这种方法提供一种功能强大的机制，通过这种机制可以根据是否定义了某个符号来包括或省略方法调用。
“委托”示例	说明如何声明、映射和组合委托。
“事件”示例	说明如何在 C# 中使用事件。
“显式接口实现”示例	阐释如何显式实现接口成员。
Hello World 示例	Hello World 应用程序。
“索引器”示例	说明如何使用数组表示法来访问对象。
“索引属性”示例	说明如何实现使用索引属性的类。索引属性使您可以使用表示类似于数组的若干种不同事物的集合的类。
“属性”示例	说明如何声明和使用属性；同时阐释抽象属性。
“结构”示例	说明如何在 C# 中使用 structs 。
“运算符重载”示例	说明用户定义的类如何能够重载运算符。
“用户定义的转换”示例	说明如何定义用户定义的类型与其他类型之间的转换。

“版本控制”示例	通过使用 override 和 new 关键字演示 C# 中的版本控制。
“Yield”示例	阐释用于筛选集合中的项的 yield 关键字。

中级示例和高级示例

“属性”示例	说明如何创建自定义属性类、如何在代码中使用这些类以及如何通过反射查询它们。
“COM Interop 第一部分”示例	说明如何使用 C# 与 COM 对象交互操作。
“COM Interop 第二部分”示例	说明如何将 C# 服务器与 C++ COM 客户端程序一起使用。
“库”示例	说明如何使用编译器选项从多个源文件来创建 DLL；同时也说明如何在其他程序中使用该库。
可空示例	演示可以设置为空的值类型。
OLE DB 示例	说明如何在 C# 中使用 Microsoft Access 数据库。它显示如何创建数据集并从数据库向该数据集添加表。
分部类型示例	演示如何在多个 C# 源代码文件中定义类和结构。
“平台调用”示例	说明如何从 C# 中调用导出的 DLL 函数。
“安全”示例	讨论 .NET Framework 安全性并说明 C# 中的两种修改安全权限的方法：使用权限类和权限属性。
“线程”示例	说明各种线程活动，如创建和执行线程、同步线程、在线程间交互以及使用线程池等。
“不安全代码”示例	说明如何使用指针。
“XML 文档”示例	说明如何使用 XML 将代码编入文档。

相关章节

[示例与快速入门](#) | [Visual C# 演练](#)

Hello World 示例

[Download sample](#)

本示例说明用 C# 编写的 Hello World 程序的几个版本。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开示例文件

1. 单击“下载示例”。
2. 出现“文件下载”消息框。
3. 单击“打开”，并在 Zip 文件夹的左列单击“提取所有文件”。
4. “提取向导”打开。
5. 单击“下一步”。您可以更改将文件提取到的目录，或再次单击“下一步”。
6. 请确保选中了“显示提取的文件”复选框，并单击“完成”。
7. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明项目解决方案不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“Hello World”示例

1. 打开解决方案 (HelloWorld.sln)。
2. 在“解决方案资源管理器”中，右击“HelloWorld1”项目并单击“设为启动项目”。
3. 在“调试”菜单上单击“开始执行(不调试)”。
4. 按任一键关闭“HelloWorld1”。
5. 在“解决方案资源管理器”中，右击“HelloWorld2”项目并单击“设为启动项目”。
6. 在“调试”菜单上单击“开始执行(不调试)”。
7. 按任一键关闭“HelloWorld2”。
8. 在“解决方案资源管理器”中，右击“HelloWorld3”项目并单击“设为启动项目”。
9. 在“解决方案资源管理器”中，右击“HelloWorld3”项目并单击“属性”。
10. 打开“配置属性”文件夹并单击“调试”。
11. 在“命令行参数”属性中键入 **A B C D**，然后单击“确定”。
12. 在“调试”菜单上单击“开始执行(不调试)”。
13. 按任一键关闭“HelloWorld3”。
14. 在“解决方案资源管理器”中，右击“HelloWorld4”项目并单击“设为启动项目”。
15. 在“调试”菜单上单击“开始执行(不调试)”。
16. 按任一键关闭“HelloWorld4”。

从命令行生成并运行“Hello World”示例

1. 使用“更改目录”命令转到“HelloWorld”目录。
2. 键入下列内容：

```
cd HelloWorld1  
csc Hello1.cs  
Hello1
```

3. 键入下列内容:

```
cd ..\HelloWorld2  
csc Hello2.cs  
Hello2
```

4. 键入下列内容:

```
cd ..\HelloWorld3  
csc Hello3.cs  
Hello3 A B C D
```

5. 键入下列内容:

```
cd ..\HelloWorld4  
csc Hello4.cs  
Hello4
```

请参见

概念

[Hello World -- 您的第一个程序 \(C# 编程指南\)](#)

C# 编程指南

其他资源

[Visual C# 示例](#)

“命令行参数”示例

[Download sample](#)

本示例说明如何访问命令行，并说明访问命令行参数数组的两种方法。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“命令行参数”示例

- 在“解决方案资源管理器”中，右击“CmdLine1”项目并单击“设为启动项目”。
- 在“解决方案资源管理器”中，右击此项目并单击“属性”。
- 打开“配置属性”文件夹并单击“调试”。
- 在“命令行参数”属性中，键入命令行参数并单击“确定”。(有关示例，请参见教程。)
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 CmdLine2 的步骤。

从命令行生成并运行“命令行参数”示例

- 使用“更改目录”命令转到“CmdLine1”目录。
- 键入下列内容：

```
csc cmdline1.cs
cmdline1 A B C
```

- 使用“更改目录”命令转到“CmdLine2”目录。
- 键入下列内容：

```
csc cmdline2.cs
cmdline2 John Paul Mary
```

请参见

任务

[如何：显示命令行参数 \(C# 编程指南\)](#)

[如何：使用 foreach 访问命令行参数 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“数组”示例

[Download sample](#)

本示例描述和说明在 C# 中数组是如何工作的。有关更多信息, 请参见[数组 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念, 并不代表着最安全的编码实践, 因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害, Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”, 并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录, 然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框, 并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“数组”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“数组”示例

- 在命令提示处键入以下内容:

```
csc arrays.cs  
arrays
```

请参见

参考

[将数组作为参数传递 \(C# 编程指南\)](#)

[使用 ref 和 out 传递数组 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“属性”示例

[Download sample](#)

本示例说明属性为何是 C# 编程语言必不可少的一个组成部分。它阐释如何声明和使用属性。有关更多信息，请参见[属性](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“属性”示例

- 在“解决方案资源管理器”中，右击“Person”项目并单击“设为启动项目”。
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 shapetest 的步骤。

从命令行生成并运行“属性”示例

- 使用“更改目录”命令转到“person”目录。
- 键入下列内容：

```
csc person.cs
person
```

- 使用“更改目录”命令转到“shapetest”目录。
- 键入下列内容：

```
csc abstractshape.cs shapes.cs shapetest.cs
shapetest
```

请参见

任务

[如何：声明和使用读/写属性\(C# 编程指南\)](#)

[如何：定义抽象属性\(C# 编程指南\)](#)

参考

[属性和索引器之间的比较\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“库”示例

[Download sample](#)

本示例说明如何在 C# 中创建和使用 DLL。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“库”示例

- 在“解决方案资源管理器”中，右击“FunctionTest”项目并单击“设为启动项目”。
- 在“解决方案资源管理器”中，右击“FunctionTest”项目并单击“属性”。
- 打开“配置属性”文件夹并单击“调试”。
- 在“命令行参数”属性中，输入 3 5 10。
- 单击“确定”。
- 在“调试”菜单上单击“开始执行(不调试)”。这将自动在“Functions”目录中生成库并执行程序。

从命令行生成并运行“库”示例

- 使用“更改目录”命令转到“Functions”目录。
- 键入下列内容：

```
csc /target:library /out:Functions.dll Factorial.cs DigitCounter.cs
```

- 使用“更改目录”命令转到“FunctionTest”目录。
- 键入下列内容：

```
copy ..\Functions\Functions.dll .
csc /out:FunctionTest.exe /R:Functions.DLL FunctionClient.cs
FunctionTest 3 5 10
```

请参见

任务

[如何：创建和使用 C# DLL\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“版本控制”示例

[Download sample](#)

本示例通过使用 **override** 和 **new** 关键字来演示 C# 中的版本控制。版本控制可帮助基类和派生类在演变过程中维护它们之间的兼容性。有关其他信息，请参见[使用 Override 和 New 关键字进行版本控制\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
- 出现“文件下载”消息框。
2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。
- 请确保选中了“显示提取的文件”复选框，并单击“完成”。
4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“版本”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“版本”示例

- 在命令提示处键入以下内容：

```
csc versioning.cs  
versioning
```

[请参见](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“集合类”示例

[Download sample](#)

本示例说明如何实现可与 **foreach** 语句一起使用的集合类。有关更多信息，请参见[集合类\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“集合类”示例

1. 在解决方案资源管理器中，右击“CollectionClasses1”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 重复前面用于 CollectionClasses2 的步骤。

从命令行生成并运行“集合类”示例

1. 使用“更改目录”命令转到“CollectionClasses1”目录。
2. 键入下列内容：

```
csc tokens.cs
tokens
```

3. 使用“更改目录”命令转到“CollectionClasses2”目录。
4. 键入下列内容：

```
csc tokens2.cs
tokens2
```

请参见

任务

[如何：使用 foreach 访问集合类\(C# 编程指南\)](#)

参考

[System.Collections.Generic](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

[常用的集合类型](#)

“结构”示例

[Download sample](#)

本示例介绍结构的语法和用法。它还涉及类与结构之间的重大差异。有关更多信息，请参见[对象、类和结构\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“结构”示例

- 在“解决方案资源管理器”中，右击“Struct1”项目并单击“设为启动项目”。
- 在“调试”菜单中，单击“开始执行(不调试)”。
- 重复前面用于 Struct2 的步骤。

从命令行生成并运行“结构”示例

- 使用“更改目录”命令转到“Struct1”目录。
- 键入下列内容：

```
csc struct1.cs
struct1
```

- 使用“更改目录”命令转到“Struct2”目录。
- 键入下列内容：

```
csc struct2.cs
struct2
```

请参见

任务

[如何：了解向方法传递结构和向方法传递类引用之间的区别\(C# 编程指南\)](#)

参考

[struct\(C# 参考\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“索引器”示例

[Download sample](#)

本示例说明 C# 类如何声明索引器以提供对类的类似数组的访问。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“索引器”示例

- 在“解决方案资源管理器”中，右击“索引器”项目并单击“属性”。
- 打开“配置属性”文件夹并单击“调试”。
- 在“命令行参数”属性中，输入“..\\..\\Test.txt”。
- 单击“确定”。
- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“索引器”示例

- 若要编译示例程序，请在命令提示处键入以下内容：

```
csc indexer.cs
```

- 示例程序将反转作为命令行参数给出的文件中的字节。例如，若要反转 Test.txt 文件中的字节并查看结果，请发出下列命令：

```
indexers Test.txt
type Test.txt
```

- 若要将反转的文件更改回正常状态，请在同一文件上再次运行该程序。

请参见

参考

[使用索引器\(C# 编程指南\)](#)

[接口中的索引器\(C# 编程指南\)](#)

[属性和索引器之间的比较\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

Visual C# 示例

“索引属性”示例

[Download sample](#)

本示例说明 C# 类如何能够声明索引属性以表示不同种类事物的类似数组的集合。有关更多信息，请参见[属性\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
2. 出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
3. “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“索引属性”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“索引属性”示例

- 在命令提示处键入以下内容：

```
csc indexedproperty.cs  
indexedproperty
```

请参见

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“用户定义的转换”示例

[Download sample](#)

本示例说明如何定义到类或结构的转换或者从类和结构的转换，并说明如何使用这样的转换。有关更多信息，请参见[转换运算符 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“用户定义的转换”示例

1. 在解决方案资源管理器中，右击“Conversion1”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 重复前面用于 Conversion2 的步骤。

从命令行生成并运行“用户定义的转换”示例

1. 使用“更改目录”命令转到“Conversion1”目录。
2. 键入下列内容：

```
csc conversion.cs
conversion
```

3. 使用“更改目录”命令转到“Conversion2”目录。
4. 键入下列内容：

```
csc structconversion.cs
structconversion
```

请参见

任务

[如何：在结构间实现用户定义的转换 \(C# 编程指南\)](#)

参考

[隐式数值转换表 \(C# 参考\)](#)

[显式数值转换表 \(C# 参考\)](#)

概念

[C# 编程指南](#)

[其他资源](#)

Visual C# 示例

“泛型”示例 (C#)

[Download sample](#)

此示例说明如何创建一个具有单个类型参数的自定义泛型列表类，以及如何实现 **IEnumerable<T>**，对列表的内容启用 **foreach** 迭代。此示例还说明客户端代码如何通过指定类型参数来创建该类的实例，以及该类型参数的约束如何实现在类型参数上执行附加操作。

有关实现迭代器块的泛型集合类的示例，请参见[如何：为泛型列表创建迭代器块 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
- 出现“文件下载”消息框。
2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“泛型”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“泛型”示例

- 在命令提示处键入以下内容：

```
csc generics.cs
generics
```

注释

提供此示例是出于演示目的，必须经过修改后才能在生产代码中使用。为获得成品质量的代码，强烈建议您尽可能使用 [System.Collections.Generic](#) 命名空间中的集合类。

请参见

参考

[System.Collections.Generic](#)

概念

[泛型 \(C# 编程指南\)](#)

其他资源

[Visual C# 示例](#)

“运算符重载”示例

[Download sample](#)

本示例说明用户定义的类如何能够重载运算符。有关更多信息，请参见 [C# 运算符](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“运算符重载”示例

- 在“解决方案资源管理器”中，右击“Complex”项目并单击“设为启动项目”。
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 Dbbool 的步骤。

从命令行生成并运行“运算符重载”示例

- 使用“更改目录”命令转到“Complex”目录。
- 键入下列内容：

```
csc complex.cs
complex
```

- 使用“更改目录”命令转到“Dbbool”目录。
- 键入下列内容：

```
csc dbbool.cs
dbbool
```

请参见

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“委托”示例

[Download sample](#)

本示例说明委托类型。示例演示了如何将委托映射到静态方法和实例方法，以及如何组合它们创建多路广播委托。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
2. 出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
3. “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“委托”示例

1. 在“解决方案资源管理器”中，右击“Delegates1”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 重复前面用于 Delegates2 的步骤。

从命令行生成并运行“委托”示例

1. 使用“更改目录”命令转到“Delegates1”目录。
2. 键入下列内容：

```
csc bookstore.cs  
bookstore
```

3. 使用“更改目录”命令转到“Delegates2”目录。
4. 键入下列内容：

```
csc compose.cs  
compose
```

请参见

任务

[如何：合并委托（多路广播委托）\(C# 编程指南\)](#)

参考

[使用委托\(C# 编程指南\)](#)

概念

[委托\(C# 编程指南\)](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“事件”示例

[Download sample](#)

本示例说明如何在 C# 中声明、调用和配置事件。有关更多信息，请参见[事件 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“事件”示例

- 在“解决方案资源管理器”中，右击“Events1”项目并单击“设为启动项目”。
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 Events2 的步骤。

从命令行生成并运行“事件”示例

- 使用“更改目录”命令转到“Events1”目录。
- 键入下列内容：

```
csc events1.cs
events1
```

- 使用“更改目录”命令转到“Events2”目录。
- 键入下列内容：

```
csc events2.cs
events2
```

请参见

任务

- [如何：订阅和取消订阅事件 \(C# 编程指南\)](#)
- [如何：发布符合 .NET Framework 准则的事件 \(C# 编程指南\)](#)
- [如何：引发派生类中的基类事件 \(C# 编程指南\)](#)
- [如何：实现接口事件 \(C# 编程指南\)](#)
- [如何：使用字典存储事件实例 \(C# 编程指南\)](#)

概念

- [C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“显式接口实现”示例

[Download sample](#)

本示例说明如何显式地实现接口成员以及如何从接口实例访问这些成员。有关背景信息，请参见[接口 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“显式接口实现”示例

- 在解决方案资源管理器中，右击“ExplicitInterface1”项目并单击“设为启动项目”。
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 ExplicitInterface2 的步骤。

从命令行生成并运行“显式接口实现”示例

- 使用“更改目录”命令转到“ExplicitInterface1”目录。
- 键入下列内容：

```
csc explicit1.cs
explicit1
```

- 使用“更改目录”命令转到“ExplicitInterface2”目录。
- 键入下列内容：

```
csc explicit2.cs
explicit2
```

请参见

任务

[如何：显式实现接口成员 \(C# 编程指南\)](#)

[如何：使用继承显式实现接口成员 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“条件方法”示例

[Download sample](#)

本示例说明条件方法，它们提供一种功能强大的机制，通过它可以根据是否定义了符号来包括或省略方法调用。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“条件方法”示例

- 在解决方案资源管理器中，右击此项目并单击“属性”。
- 打开“配置属性”文件夹并单击“调试”。
- 将“命令行参数”属性设置为“**A B C**”。
- 在“配置属性”文件夹中，单击“生成”。
- 修改“条件编译常量”属性。例如，添加或删除 DEBUG。单击“确定”。
- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“条件方法”示例

- 若要包括条件方法，则通过在命令提示处键入以下内容来编译和运行此示例程序：

```
csc CondMethod.cs tracetest.cs /d:DEBUG
tracetest A B C
```

请参见

参考

[Conditional\(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“XML 文档”示例

[Download sample](#)

此示例说明如何使用 XML 将代码编入文档。有关附加信息，请参见 [XML 文档注释 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
2. 出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
3. “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成“XML 文档”示例

1. 在解决方案资源管理器中，右击此项目并单击“属性”。
2. 打开“配置属性”文件夹并单击“生成”。
3. 将“XML 文档文件”属性设置为 XMLsample.xml。
4. 在“生成”菜单上单击“生成”。XML 输出文件将出现在调试目录中。

从命令行生成“XML 文档”示例

1. 若要生成示例 XML 文档，请在命令提示处键入下列内容：

```
csc XMLsample.cs /doc:XMLsample.xml
```

2. 若要查看生成的 XML，请发出下列命令：

```
type XMLsample.xml
```

请参见

任务

[如何：使用 XML 文档功能 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“平台调用”示例

[Download sample](#)

本示例说明如何从 C# 中调用平台调用(导出的 DLL 函数)。有关更多信息, 请参见[互操作性\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念, 并不代表着最安全的编码实践, 因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害, Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
2. 单击“打开”, 并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录, 然后再单击“下一步”。
4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“平台调用”示例

1. 在解决方案资源管理器中, 右击“PInvokeTest”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 重复前面用于 Marshal 和 PInvoke 的步骤。

从命令行生成并运行“平台调用”示例

1. 使用“更改目录”命令转到“PInvokeTest”目录。
2. 键入下列内容:

```
csc PInvokeTest.cs
PInvokeTest
```

3. 使用“更改目录”命令转到“Marshal”目录。
4. 键入下列内容:

```
csc Marshal.cs
Marshal
```

5. 使用“更改目录”命令转到“PInvoke”目录。
6. 键入下列内容:

```
csc logfont.cs pinvoke.cs
pinvoke
```

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“COM Interop 第一部分”示例

[Download sample](#)

本示例说明 C# 程序如何能够与非托管 COM 组件交互操作。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。Microsoft 并不保证将该示例代码用于除此以外的其他用途时不会造成意外或连带损坏。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。
- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“COM Interop 第一部分”示例

- 在“解决方案资源管理器”中，右击“Interop1”项目并单击“设为启动项目”。
- 在“解决方案资源管理器”中，右击“Interop1”项目并单击“属性”。
- 打开“配置属性”文件夹并单击“调试”。
- 在“命令行参数”属性中输入一个 AVI 文件，例如 c:\winnt\clock.avi。
- 单击“确定”。
- 在“调试”菜单上，单击“开始执行(不调试)”。
- 重复前面用于 Interop2 的步骤。

如果 Interop1 中包含的 QuartzTypeLib.dll 已过时

- 在“解决方案资源管理器”中，为 Interop1 打开“引用”。
- 右击“QuartzTypeLib”并单击“移除”。
- 右击“引用”并单击“添加引用”。
- 在“COM”选项卡中，选择名为“ActiveMovie 控件类型库”的组件。
- 单击“选择”，然后单击“确定”。
- 重新生成 Interop1。

注意

向组件中添加引用与在命令行中调用 tlbimp 以创建 QuartzTypeLib.dll（如下所述）的效果相同。

从命令行生成并运行“COM Interop 第一部分”示例

- 使用“更改目录”命令转到“Interop1”目录。

2. 键入下列内容：

```
tlbimp %windir%\system32\quartz.dll /out:QuartzTypeLib.dll  
csc /r:QuartzTypeLib.dll interop1.cs  
interop1 %windir%\clock.avi
```

3. 使用“更改目录”命令转到“Interop2”目录。

4. 键入下列内容：

```
csc interop2.cs  
interop2 %windir%\clock.avi
```

[请参见](#)

[参考](#)

[互操作性\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“COM Interop 第二部分”示例

[Download sample](#)

本示例说明如何将 C# 服务器与 C++ COM 客户端程序一起使用。

注意

必须安装 Visual C++ 才能编译此示例。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“COM Interop 第二部分”示例

1. 在“解决方案资源管理器”中，右击“COMClient”项目并单击“属性”。
2. 打开“配置属性”文件夹并单击“调试”。
3. 在“命令行参数”属性中，输入名称。
4. 单击“确定”。
5. 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“COM Interop 第二部分”示例

1. 使用“更改目录”命令转到“COMInteropPart2\COMClient”目录。
2. 将 C# 服务器代码复制到“COMClient”目录中：

```
copy ..\CSharpServer\CSharpServer.cs
```

3. 编译服务器：

```
csc /target:library CSharpServer.cs
regasm CSharpServer.dll /tlb:CSharpServer.tlb
```

4. 编译客户端程序(确保在 vcvars32.bat 中正确设置了路径和环境变量)：

```
cl COMClient.cpp
```

5. 运行客户端程序：

```
COMClient friend
```

[请参见](#)

[参考](#)

[互操作性\(C# 编程指南\)](#)

[概念](#)

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“属性”示例

[Download sample](#)

本示例说明如何创建自定义属性类，如何在代码中使用这些类，以及如何通过反射查询它们。有关属性的更多信息，请参见[属性 \(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。

出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“属性”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“属性”示例

- 在命令提示处键入以下内容：

```
csc AttributesTutorial.cs
AttributesTutorial
```

请参见

任务

[如何：使用属性创建 C/C++ 联合 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“安全”示例

[Download sample](#)

本示例阐释如何通过权限类和权限属性来修改安全权限。有关其他信息，请参见[安全性\(C# 编程指南\)](#)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。
- 出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
- “提取向导”打开。

- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。
- 请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“安全”示例

- 在解决方案资源管理器中，右击“Security1”项目并单击“设为启动项目”。
- 在“调试”菜单上单击“开始执行(不调试)”。
- 重复前面用于 Security2 和 Security3 的步骤。

从命令行生成并运行“安全”示例

- 使用“更改目录”命令转到“Security1”目录。
- 键入下列内容：

```
csc ImperativeSecurity.cs
ImperativeSecurity
```

- 使用“更改目录”命令转到“Security2”目录。
- 键入下列内容：

```
csc DeclarativeSecurity.cs
DeclarativeSecurity
```

- 使用“更改目录”命令转到“Security3”目录。
- 键入下列内容：

```
csc SuppressSecurity.cs
SuppressSecurity
```

请参见 概念

[C# 编程指南](#)

[其他资源](#)

[Visual C# 示例](#)

“线程”示例

[Download sample](#)

此示例演示下面的线程处理技术。有关更多信息，请参见[线程处理\(C# 编程指南\)](#)。

- 创建、启动和终止线程
- 使用线程池
- 线程同步和互交

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

 2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

 3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

 4. 双击该示例的 .sln 文件。
- 示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“线程”示例

1. 在“解决方案资源管理器”中，右击“ThreadStartStop”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 重复前面用于 ThreadPool 和 ThreadSync 的步骤。

从命令行生成并运行“线程”示例

1. 使用“更改目录”命令转到“Threads”目录。
2. 键入下列内容：

```
cd ThreadStartStop
csc ThreadStartStop.cs
ThreadStartStop
```

3. 键入下列内容：

```
cd ..\ThreadPool
csc ThreadPool.cs
ThreadPool
```

4. 键入下列内容：

```
cd ..\ThreadSync
csc ThreadSync.cs
```

请参见

任务

[如何: 创建和终止线程 \(C# 编程指南\)](#)

[如何: 使用线程池 \(C# 编程指南\)](#)

[如何: 对制造者线程和使用者线程进行同步 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

“不安全代码”示例

[Download sample](#)

本示例说明如何在 C# 中使用非托管代码(使用指针的代码)。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
2. 出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
3. “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。
4. 请确保选中了“显示提取的文件”复选框，并单击“完成”。
4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“不安全代码”示例

1. 在“解决方案资源管理器”中，右击“FastCopy”项目并单击“设为启动项目”。
2. 在“调试”菜单上单击“开始执行(不调试)”。
3. 在“解决方案资源管理器”中，右击“ReadFile”项目并单击“设为启动项目”。
4. 在“解决方案资源管理器”中，右击“ReadFile”项目并单击“属性”。
5. 打开“配置属性”文件夹并单击“调试”。
6. 在“命令行参数”属性中，输入 ..\..\ReadFile.cs。
7. 单击“确定”。
8. 在“调试”菜单上单击“开始执行(不调试)”。
9. 在“解决方案资源管理器”中，右击“PrintVersion”项目并单击“设为启动项目”。
10. 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“不安全代码”示例

1. 使用“更改目录”命令转到“Unsafe”目录。
2. 键入下列内容：

```
cd FastCopy
csc FastCopy.cs /unsafe
FastCopy
```

3. 键入下列内容：

```
cd ..\ReadFile
csc ReadFile.cs /unsafe
```

4. 键入下列内容：

```
cd ..\PrintVersion  
csc PrintVersion.cs /unsafe  
PrintVersion
```

请参见

任务

[如何：使用指针复制字节数组 \(C# 编程指南\)](#)

[如何：使用 Windows ReadFile 函数 \(C# 编程指南\)](#)

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

OLE DB 示例

[Download sample](#)

本示例说明如何在 C# 中使用 Microsoft Access 数据库。它显示如何创建数据集并从数据库向该数据集添加表。示例程序中使用的 BugTypes.mdb 数据库是一个 Microsoft Access 2000 .mdb 文件。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“OLE DB”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

注意

如果您是在“发布”模式下生成解决方案，请将“BugTypes.mdb”复制到 \bin\release 文件夹中。

从命令行生成并运行“OLE DB”示例

- 在命令提示处键入以下内容：

```
csc oledbsample.cs
oledbsample
```

请参见

概念

[C# 编程指南](#)

其他资源

[Visual C# 示例](#)

[访问数据 \(Visual Studio\)](#)

“Yield”示例

[Download sample](#)

此示例说明如何创建一个列表类来实现 **IEnumerable<int>** 和 **yield** 关键字，以对列表的内容启用 **foreach** 迭代。定义了两个属性，一个返回奇数，另一个返回偶数。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。

出现“文件下载”消息框。

2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

4. 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“Yield”代码示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“Yield”代码示例

1. 使用 **Change Directory (cd)** 命令转到 **Yield** 目录。
2. 键入下列内容：

```
csc Yield.cs
Yield
```

请参见

参考

[yield\(C# 参考\)](#)

概念

[迭代器\(C# 编程指南\)](#)

其他资源

[Visual C# 示例](#)

[“泛型”示例 \(C#\)](#)

“匿名委托”示例

[Download sample](#)

此示例演示了如何使用匿名委托来计算员工的薪水奖金。使用匿名委托简化了程序，因为无需再定义一个单独的方法。

每个员工的数据都存储在一个对象中，该对象中包含了个人的详细信息和一个引用了计算奖金所需算法的委托。通过以委托的方式定义算法，可以使用同样的方法来执行奖金计算，而与实际如何计算无关。另外需要注意的是，一个局部变量 `multiplier` 变成了已捕获的外部变量，因为它在委托计算中被引用了。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。

出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“匿名委托代码”示例

- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行“匿名委托代码”示例

- 使用“更改目录 (cd)”命令转到“AnonymousDelegates”目录。
- 键入下列内容：

```
csc AnonymousDelegates.cs
AnonymousDelegates
```

请参见

参考

[匿名方法\(C# 编程指南\)](#)

概念

[委托\(C# 编程指南\)](#)

其他资源

[Visual C# 示例](#)

[C# 参考](#)

分部类型示例

[Download sample](#)

本示例演示如何使用允许在两个或更多 C# 文件中定义类或结构的分部类型。这就允许多个程序员并行处理一个类的不同部分，并将复杂类的不同方面保存在不同的文件中。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

1. 单击“下载示例”。
 出现“文件下载”消息框。
2. 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。
 “提取向导”打开。
3. 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。
 请确保选中了“显示提取的文件”复选框，并单击“完成”。
4. 双击该示例的 .sln 文件。
 示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行“分部类型代码”示例

- 在“调试”菜单中，单击“开始执行(不调试)”。

从命令行生成并运行“分部类型代码”示例

1. 使用 **Change Directory (cd)** 命令转到 **PartialTypes** 目录。
2. 键入下列内容：

```
csc PartialTypes.cs  
PartialTypes
```

请参见

参考

[partial\(C# 参考\)](#)

[分部类定义\(C# 编程指南\)](#)

其他资源

[Visual C# 示例](#)

可空示例

[Download sample](#)

本示例说明如何使用可空类型。此功能允许将值类型设置为未初始化状态(即空状态)，这与将引用类型设置为 `null` 的方式相似。

安全注意

提供该示例代码是为了阐释一个概念，并不代表着最安全的编码实践，因此不应在应用程序或网站中使用该示例代码。对于因将示例代码用于其他用途而出现的偶然或必然的损害，Microsoft 不承担任何责任。

在“解决方案资源管理器”中打开该示例文件

- 单击“下载示例”。

出现“文件下载”消息框。

- 单击“打开”，并在 Zip 文件夹窗口的左列单击“提取所有文件”。

“提取向导”打开。

- 单击“下一步”。您可以更改文件将被提取到的目录，然后再单击“下一步”。

请确保选中了“显示提取的文件”复选框，并单击“完成”。

- 双击该示例的 .sln 文件。

示例解决方案显示在“解决方案资源管理器”中。您可能会收到说明解决方案位置不受信任的安全警告。单击“确定”继续。

在 Visual Studio 中生成并运行可空代码示例

- 可通过双击 Windows 资源管理器中的文件，或单击“文件”菜单上的“打开”，打开 Nullable.sln 解决方案文件。
- 在“调试”菜单上单击“开始执行(不调试)”。

从命令行生成并运行可空代码示例

- 使用 **Change Directory (cd)** 命令转到 Nullable 目录下的 \Boxing、\Basics 或 \Operator 子目录。
- 对于“装箱”示例，请键入以下内容：

```
csc Boxing.cs
Boxing
```

对于“基本”示例，请键入以下内容：

```
csc Basics.cs
Basics
```

对于“运算符”示例，请键入以下内容：

```
csc Operator.cs
Operator
```

请参见

概念

[可空类型\(C# 编程指南\)](#)

其他资源

[Visual C# 示例](#)

