**异常和异常处理（C# 编程指南）**

C# 语言的异常处理功能可帮助您处理程序运行时出现的任何意外或异常情况。异常处理使用 **try**、**catch** 和 **finally** 关键字尝试某些操作，以处理失败情况，尽管这些操作有可能失败，但如果您确定需要这样做，且希望在事后清理资源，就可以尝试这样做。公共语言运行时 (CLR)、.NET Framework 或任何第三方库或者应用程序代码都可以生成异常。异常是使用 **throw** 关键字创建的。

很多情况下，异常可能不是由代码直接调用的方法引发，而是由调用堆栈中位置更靠下的另一个方法所引发。在这种情况下，CLR 将展开堆栈，查找是否有方法包含针对该特定异常类型的 **catch** 块，如果找到这样的方法，就会执行找到的第一个这样的 **catch** 块。如果在调用堆栈中的任何位置都没有找到适当的 **catch** 块，就会终止该进程，并向用户显示一条消息。

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。

class ExceptionTest

{

static double SafeDivision(double x, double y)

{

if (y == 0)

throw new System.DivideByZeroException();

return x / y;

}

static void Main()

{

// Input for test purposes. Change the values to see

// exception handling behavior.

double a = 98, b = 0;

double result = 0;

try

{

result = SafeDivision(a, b);

Console.WriteLine("{0} divided by {1} = {2}", a, b, result);

}

catch (DivideByZeroException e)

{

Console.WriteLine("Attempted divide by zero.");

}

}

}

**异常概述**

异常具有以下特点：

* 各种类型的异常最终都是由 **System.Exception** 派生而来。
* 在可能引发异常的语句周围使用 **try** 块。
* 一旦 **try** 块中发生异常，控制流将跳转到第一个关联的异常处理程序（无论该处理程序存在于调用堆栈中的什么位置）。在 C# 中，**catch** 关键字用于定义异常处理程序。
* 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。
* 除非您可以处理某个异常并使应用程序处于已知状态，否则请不要捕捉该异常。如果捕捉 **System.Exception**，请在 **catch** 块的末尾使用 **throw** 关键字再次引发该异常。
* 如果 **catch** 块定义了一个异常变量，则可以用它获取有关所发生异常类型的更多信息。
* 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，比如调用堆栈的状态以及有关错误的文本说明。
* 即使发生异常也会执行 **finally** 块中的代码。使用 **finally** 块释放资源，例如，关闭在 **try** 块中打开的任何流或文件。
* .NET Framework 中的托管异常是凭借 Win32 结构化异常处理机制实现的。有关更多信息，请参见[结构化异常处理 (C/C++)](https://msdn.microsoft.com/zh-cn/library/swezty51.aspx) 和 [A Crash Course on the Depths of Win32 Structured Exception Handling](http://go.microsoft.com/fwlink/?LinkId=119654)（有关深入探究 Win32 结构化异常处理的应急课程）。

# 如何：使用 try/catch 处理异常（C# 编程指南）

[try-catch](https://msdn.microsoft.com/zh-cn/library/0yd65esw.aspx) 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，解决问题后不会再次引发异常；但更多情况下，您唯一能做的是确保引发适当的异常。

## 示例

在此示例中，[IndexOutOfRangeException](https://msdn.microsoft.com/zh-cn/library/system.indexoutofrangeexception.aspx) 不是最适当的异常：对本方法而言 [ArgumentOutOfRangeException](https://msdn.microsoft.com/zh-cn/library/system.argumentoutofrangeexception.aspx) 更恰当些，因为错误是由调用方传入的 index 参数导致的。

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

导致异常的代码被括在 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](https://msdn.microsoft.com/zh-cn/library/system.exception.innerexception.aspx)。 因为 [InnerException](https://msdn.microsoft.com/zh-cn/library/system.exception.innerexception.aspx) 属性是[只读](https://msdn.microsoft.com/zh-cn/library/acdd6hb7.aspx)，所以必须在新异常的构造函数中为其赋值。

# 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（通常是保存外部资源的对象）清理。此类清理功能的一个示例是在使用后立即对 [FileStream](https://msdn.microsoft.com/zh-cn/library/system.io.filestream.aspx) 调用 [Close](https://msdn.microsoft.com/zh-cn/library/system.io.stream.close.aspx)，而不是等待公共语言运行时对该对象进行垃圾回收，如下所示：

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();

}

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。

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.UnauthorizedAccessException 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尝试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](https://msdn.microsoft.com/zh-cn/library/system.io.stream.close.aspx) 方法前，[FileStream](https://msdn.microsoft.com/zh-cn/library/system.io.filestream.aspx) 对象不为 **null**。如果没有 **null** 检查，**finally** 块可能引发自身的 [NullReferenceException](https://msdn.microsoft.com/zh-cn/library/system.nullreferenceexception.aspx)，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，所以应尽快关闭数据库连接。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。