**使用异常（C# 编程指南）**

在 C# 中，程序中的运行时错误通过使用一种称为“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 公共语言运行时 (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](https://msdn.microsoft.com/zh-cn/library/system.exception.aspx) 派生的类表示。 此类标识异常的类型，并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：

class CustomException : Exception

{

public CustomException(string message)

{

}

}

private static void TestThrow()

{

CustomException ex =

new CustomException("Custom exception in TestThrow()");

throw ex;

}

在引发异常之后，运行时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 块的类型与异常或异常的基类的类型相同，则该 catch 块就能够处理该方法。 例如：

static void TestCatch()

{

try

{

TestThrow();

}

catch (CustomException 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，运行时将检查调用方法中是否有 try 语句和 catch 块。 运行时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 块之后的下一个语句。

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。 因此，在任何情况下都应该按照从最具体（或者派生程度最高）到最不具体这一顺序排列 catch 块。 例如：

static void TestCatch2()

{

System.IO.StreamWriter sw = null;

try

{

sw = new System.IO.StreamWriter(@"C:\test\test.txt");

sw.WriteLine("Hello");

}

catch (System.IO.FileNotFoundException ex)

{

// Put the more specific exception first.

System.Console.WriteLine(ex.ToString());

}

catch (System.IO.IOException ex)

{

// Put the less specific exception last.

System.Console.WriteLine(ex.ToString());

}

finally

{

sw.Close();

}

System.Console.WriteLine("Done");

}

执行 catch 块之前，运行时会检查 finally 块。 Finally 块使程序员能够清除中止的 try 块可能遗留下的任何模糊状态，或者释放任何外部资源（例如图形句柄、数据库连接或文件流），而无需等待运行时中的垃圾回收器终结这些对象。 例如：

static void TestFinally()

{

System.IO.FileStream file = null;

//Change the path to something that works on your machine.

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。 由于要执行 finally 块（即使已引发异常），前一示例中的 finally 块使得可以正确地关闭文件，从而帮助避免错误。

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](https://msdn.microsoft.com/zh-cn/library/system.typeinitializationexception.aspx)，并将原始异常分配给新异常的 [InnerException](https://msdn.microsoft.com/zh-cn/library/system.exception.innerexception.aspx) 属性。
* 如果到达线程的开头，则终止线程。